

COMPUTER SOFTWARE APPENDIX

1
2 The following computer software appendix is being deposited as part of the specification
3 of this patent application under 37 C.F.R. 1.96 as original copies from computer printout and as
4 two sheets per one patent specification page for the ease of the publication office.

```

#include "dpcagent.h" /* Our header file */

/*****
 * History:
 *
 * Version 0.1 - First Package delivery for demo use.
 * Version 0.2 - (03-15-96)
 *
 * Site ID filtering to catalog
 * delivery support
 *
 * Stats()
 *
 * ig into file and added DPCUpdateConfig()
 * Version 0.3 - (03-25-96)
 *
 * support for interactivity field in
 *
 * se... in catalog by adding minimum
 * ore committing entry.
 * Version 0.4 - (03-29-96)
 *
 * indexes to modem configuration and
 *
 * o 2400 to fix key reception bugs.
 *
 * to enable agent to continue getting
 *
 * ssful, even if agent shuts down.
 * Version 0.5 - (04-09-96)
 *
 * showing up in Cancel Download list
 *
 * tatistics halfway running (doesn't abend)
 *
 * explicit) files in catalogue
 *
 * uest/response/confirm code running but
 * it yet.
 * Version 0.6 - (04-10-96)
 *
 * nting Get MLID stats
 *
 * alStrength()
 * Version 0.7 - (04-11-96)
 *
 * igation Editor
 *
 * ength Meter
 * Version 0.8 - (04-12-96)
 *
 * alStrength to report 0 when not connected
 *
 * nting cost(explicit) file download
 *
 * View Database Entries option worked
 *
 * entire entry and is more usable.
 * Version 0.9 - (04-16-96)
 *
 */
*****

Added Community/
Added Periodic d
Added DPCGetMLID
Broke modem conf
Added SFX_PUSH s
catalog.
Fixed FreeFreeFr
length check bef
Added baud rate
defaulted them t
Also added code
keys until succe
Fixed push files
Got DisplayMLIDS
Recognized cost(
Got explicit req
NOC doesn't send
Finished impleme
Added DPCGetSign
Added Modem Conf
Added Signal Str
Fixed DPCGetSign
Finished impleme
Modified the way
so that it shows
Completed Turbo

```

Internet support. Can't totally test

```

* away is up and running.
* Version 0.10 - (04-24-96)
*
* ck
*
* lse" in IPSEndRoutine after call
*
* ts. Turbo Internet works 90%.
* Version 0.11 - (04-25-96)
*
* AppendStringField in DLO to include
*
* nstead of NULL.
* Version 0.12 - (05-16-96)
*
* ase 2.
*
* between ATDT and 1-800...
*
* r commas in phone numbers and prefix
*
* ing to accept & and -
*
* Gateway strings to have (ISP) follow it
*
* played on if -DEBUG is on command line
* Version 1.00 - (06-11-96)
*
* ud rate to Modem Status string
*
* it doesn't stay in during internet traffic
*
* formation screen
*
* er edit strings to allow alphas
*
* buffer to 4K(default was 1K)
* Version 1.01 - (06-17-96)
*
* iver comm code into driverio.c
*
* ster call to DPC.LAN so that it could
*
* removes which would allow us to prevent
*
* Version 1.02 - (06-25-96)
*
* ch to disable package delivery for debug
* Version 1.04 - (06-25-96)
*
* time to send directly to AIO.
* Version 1.05 - (06-28-96)
*
* umber in menu to decimal
*
* size field to modem configuration
*
* which changed configuration screens
* Version 1.06 - (07-05-96)
*
* te tinet fragment routines(jkt).
*
* rol menu.
*

```

```

yet until IJ gat
Added ARP loopba
Fixed missing "e
to GatherFragmen
Fixed NWSAppennd
character sets i
Completion of Ph
Places prefix in
Added support fo
Allowed init str
Modified IP and
Debug screen dis
Added connect ba
Fixed Tx LED so
Added Adapter In
Fixed phone numb
Bumped AIO write
Broke out LAN dr
Added Agent Regi
call us when it
abends.
Added -NOPD swit
Revised send rou
Changed serial n
Added AIO buffer
Added auto login
Completely rewro
Added Modem Cont
Added GNU condit

```

```
ionals.
*
pt timeouts
*
utines into PPP.C
*
*****
#define AGENT_VERSION INMSG("1.20 ", 173)
/*****
* Global variable used by DPCAGENT.C
*****
*/
/* Resource Tag variables.
*/
struct LoadDefinitionStructure *NLMHandle = 0;
struct ResourceTagStructure *allocrTag = 0;
struct ResourceTagStructure *timerTag = 0;
struct ResourceTagStructure *ABSTag = 0;
struct ResourceTagStructure *asynctioTag = 0;
/*
* NUT and screen ID variables.
*/
NUTInfo
NUT handle
/*
* NUT handle = NULL;
*/
/* if DEBUG_ALL
struct ScreenStruct *DebugScreenID = 0; /* for OutputToScreen
*/
#ifdef
WORD
/* Screen Height(25)
*/
WORD
/* Screen Width(80)
*/
LONG
ackground Portal
/*
* Process and thread variables.
*/
BYTE
inter to messages
LONG
/* Main Handler thread PID
*/
LONG
/* Packet Handler thread PID
*/
LONG
DPCModemPID = 0;
/* Modem Handler thread PID
*/
LONG
/* Access thread PID
*/
LONG
/* Turbo Internet thread PID
*/
BYTE
int
/* All threads must exit
*/
LONG
/* Tinet needs to wake up flag
*/
int
/*
* CRC table used by calccrc().
*/
static unsigned short crcTab[256] = {
0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xcac6, 0xdb55, 0xe97e, 0xf8f7,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
```

```
0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbce3, 0x8e58, 0x9fd1, 0x6fae, 0x7c87, 0x4d9f, 0x5eb5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x433c,
0xbdc8, 0xac42, 0x9ed9, 0x8f50, 0xfbf5, 0xea66, 0xd8fd, 0xc974,
0x4204, 0x338d, 0x6116, 0x0420, 0x709f, 0x615a, 0x7232, 0x36bb,
0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdec8, 0xf444, 0xfdd1, 0xec56, 0x98e9, 0x8960, 0xbfb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xc5c5, 0xdd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0xf787, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xf7cf, 0xee46, 0xdcdd, 0xcdf5, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7c7f,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7af7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8038, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0xc60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x91b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};
```

/* MLID Statistic globals */

```
int
{
GenericDescriptionTable[22*8] =
{
InMSG("Total packets sent:", 103),
InMSG("Total packets received:", 141),
InMSG("No ECB available count:", 148),
InMSG("Send packet too big count:", 156),
InMSG("Reserved:", 186),
InMSG("Receive packet overflow count:", 214),
InMSG("Receive packet too big count:", 215),
InMSG("Receive packet too small count:", 216),
InMSG("Send packet miscellaneous errors:", 217),
InMSG("Receive packet miscellaneous errors:", 218),
InMSG("Send packet retry count:", 223),
InMSG("Checksum errors:", 290),
InMSG("Hardware receive mismatch count:", 291),
InMSG("Total send OK byte count low:", 229),
InMSG("Total send OK byte count high:", 230),
InMSG("Total receive OK byte count low:", 231),
InMSG("Total receive OK byte count high:", 232),
InMSG("Total group address send count:", 233),
InMSG("Total group address receive count:", 251),
InMSG("Adapter reset count:", 252),
InMSG("Adapter operating time stamp:", 253),
InMSG("Adapter queue depth:", 255),
InMSG("Send OK single collision count:", 159),
InMSG("Send OK multiple collision count:", 256),
InMSG("Send OK but deferred", 264),
InMSG("Send abort from late collision", 265),
InMSG("Send abort from excess collision", 266),
InMSG("Send abort from carrier sense", 267),
InMSG("Send abort from excessive deferral", 268),
InMSG("Receive abort from bad frame alignment", 269)
```

```

);

#define STATS_DATA_WIDTH      13
#define FSD_DATA_WIDTH       40
#define BORDER_WIDTH         3
#define INDENT_WIDTH         3

void ( *BackgroundFuncPtr ) ( LONG portalNumber ) = NULL;
LONG BackgroundPortal;
int GenericLinesStart;
int GblDataCol;
struct DOSCountryInfoStruct GblDOSCountryInfo;

int DebugFlag = FALSE;

/* The array nDeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location is not in mainland US
 */
static float nDeclination[] =
{
    0, 0, 4, 1, 0, 0, 0, -8, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, -3, -4, -6, -8, -10, -11, 12, -13, 0, 0,
    0, 0, 5, 2, -2, -5, -6, -9, -11, -12, -13, -13, -14,
    0, 7, 5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16,
    0, 12, 9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17,
    18, 15, 0, 7, 2, -3, -6, -9, -11, -13, -15, -18, -19,
    0, 0, 0, 0, 0, -1, -6, -9, -13, -15, -18, -20, -21, -22
};

#ifdef LOG_ECB_ACTIVITY
int DPC_TGID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*
 * Local function prototypes.
 */
void ReturnResources(int sig);
ExitHandler(void *handle)

Description:
    This routine is called when the user hits Alt-F10 to exit
    the utility. We will attempt to verify with the user that
    they really want to exit.

Input:    handle        - NUT handle

Output:    nothing

Returns:    nothing

```

```

ReturnResources(1);
exit(1);
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
    head = head;
    tail = tail;
    handle = handle;
    return;
}

/*****
 * calccrc(WORD crc,
 *         BYTE *cp,
 *         LONG len)
 *
 * Description:
 *     This routine calculates a CRC value for the text passed
 *     in. It uses a CRC substitution table for efficiency.
 *     Its used by the Slip transmit routine.
 *
 * Input:    crc        - Initial value of the C
 *             cp         - string to calc
 *             len        - length of the string
 *
 * Output:    nothing
 *
 * Returns:   16-bit CRC value
 */
WORD calccrc(WORD crc, BYTE *cp, LONG len)
{
    while(len--)
        crc = (crc >> 8) ^ crctab[(crc ^ *cp++) & 0xff];
    return(crc);
}

/*****
 * SlipSend(char *pdata,
 *          LONG sz,
 *          int cas,
 *          int timeout)
 *
 * Description:
 *     This routine builds a SLIP envelope around the message
 *     passed in, escapes any HDLC characters in the message,
 *     and sends it to the modem.
 *
 * Input:    pdata        - Pointer to the message
 *             sz          - length of the
 *             message passed in
 */

```



```

* send to hub
*
* timeout
*
* Output: nothing
*
* Returns: nothing
*
* *****
void SlipSend( char *pdata, LONG sz, int cas, int timeout)
(
    LroEspkt_t txcas;
    LroAspkt_t txhub;
    LONG tmlen;
    WORD crc;
    BYTE *pi, *po;
    BYTE slip_data[3072];
    int is, os;

    if (cas)
    (
        /* initialize cas transfer */
        CMovB(pdata, txcas.data, sz);
        txcas.length = sz + (txcas.data - ((BYTE *)txcas));
        tmlen = txcas.length;
        txcas.length |= 0x8000;
        crc = calccrc(INITCRC, (BYTE *)txcas, tmlen);
        txcas.data[sz] = crc & 0xff;
        txcas.data[sz+1] = (crc >> 8) & 0xff;
        pi = (BYTE *)txcas;
    )
    else
    (
        /* initialize hub transfer */
        CMovB(pdata, txhub.data, sz);
        txhub.length = sz + (txhub.data - ((BYTE *)txhub));
        txhub.filler1 = txhub.channel = 0;
        txhub.control = 0x8000;
        tmlen = txhub.length;
        crc = calccrc(INITCRC, (BYTE *)txhub, tmlen);
        txhub.data[sz] = crc & 0xff;
        txhub.data[sz+1] = (crc >> 8) & 0xff;
        pi = (BYTE *)txhub;
    )

    /* Start out with SLIP start character */
    po = slip_data;
    *po++ = END;

    /* Escape any special SLIP characters that may be in the message */
    for (is = 0, os = 1; is < (tmlen + 2); is++, os++, pi++, po++)
    (
        switch(*pi)
        (
            case END:
                *po++ = ESC;
                *po = ESC_END;
                os++;
                break;
            case ESC:
                *po++ = ESC;
                *po = ESC_ESC;
                os++;
        )
    )
}

```

```

break;
default:
    *po = *pi;
    break;
}

/* Finish packet off with SLIP END character */
*po = END;
os++;

/* Send message to the modem thread */
DloSend(slip_data, os, timeout);
)

/*****
*
* EXPORTED FUNCTION
*
* DPCGetBoard(LONG *board,
*             LONG *controlEntry)
*
* Description:
*   This routine returns the DPC MLID logical board number
*   and control entry address to be used to send IOCTL
*   requests to the DPC mlid. The ioctlMlid pragma can
*   then be used to make the request.
*
* Input:
*   board
*   ore board number
*   controlEntry
*   y address
*
* Output:
*   board and controlEntry filled in if successful
*
* Returns:
*   0 if MLID is active
*
* *****
LONG DPCGetBoard(LONG *board, LONG *controlEntry)
(
    if (DIOBoard == 0)
        return(-1);

    *board = DIOBoard;
    *controlEntry = DIOControlEntry;
    return(0);
)

void
CreateStringWithCommas( LONG number, BYTE *buffer, char *format )
(
    int i;
    int j;
    int found;
    int length;
    int commasAdded = 0;
    BYTE tmpBuf[ 128 ];
    *tmpPtr;

    NWSprintf( tmpBuf, format, number);
    for ( i = ( length = CStrLen( tmpBuf ) ), found = 0; i >= 0; i-- )

```

```

(
    if ( ( tmpBuf[ i ] >= '0' ) && ( tmpBuf[ i ] <= '9' ) )
    {
        /* we have a digit */
        if ( ++found > 3 )
        {
            found = 1;

            /* shift the string one to the right */
            for ( j = ++length; j > i; j-- )
                tmpBuf[ j ] = tmpBuf[ j - 1 ];

            tmpBuf[ i + 1 ] = GblDOSCountryInfo.thousandSep[ 0 ];
            commasAdded++;
        }
    }

    /* Adjust the length of the string back to its original if there are
    ** leading spaces.
    */
    for ( i = 0, tmpPtr = tmpBuf; i < commasAdded; i++ )
    {
        if ( *tmpPtr == ' ' )
            tmpPtr++;
    }
    CStrCpy( buffer, tmpPtr );
}

void
SecondsToDateAndTime( LONG seconds, BYTE *buff )
{
    char *ascBuf;

    ascBuf = asctime( localtime( (time_t *) &seconds ) );
    CHovB( ascBuf, buff, 26 );
    buff[ 24 ] = 0;
}

void
FormatElapsedTime( LONG seconds, LONG tenths, BYTE *buff )
{
    LONG minutes, hours, days;

    /* convert secs to minutes */
    minutes = seconds / 60;
    seconds = seconds % 60;
    hours = minutes / 60;
    minutes = minutes % 60;
    days = hours / 24;
    hours = hours % 24;
    if ( days > 0 )
    {
        NWSprintf
        (
            buff,
            MSG( "%d %c%02d%c%02d%c%01d", 293 ),
            days,
            GblDOSCountryInfo.timeSep[ 0 ],
            hours,
            GblDOSCountryInfo.timeSep[ 0 ],
            minutes,
            GblDOSCountryInfo.timeSep[ 0 ],
            seconds,
            GblDOSCountryInfo.timeSep[ 0 ],
            tenths
        );
    }
}

void
HandleScrollablePortal( PCB *portal )
{
    int escapeFlag = 0;
    LONG keyType;
    BYTE ch;
    int updatedDisplay = TRUE;
    LONG virtualHeight;
    LONG portalHeight;
    LONG bottomLine;
    LONG curPos;
    LONG vLine;

    /*
    ** The values for portal->portalHeight and portal->virtualHeight are the
    ** only two that we deal with that are 1-based. All the rest are 0-based
    */
}

```

```

h
** so we will use local copies of these variables adjusted to fit in wit
** the rest in the PCB structure.
**/
virtualHeight = portal->virtualHeight - 1;
portalHeight = portal->portalHeight - 1;
/*
** Other initializations.
**/
bottomLine = virtualHeight - portalHeight;

while(!escapeFlag)
(
    if ( updatedDisplay == TRUE )
    {
        /*
        ** The last iteration of the loop made a change, so redr
        ** portal.
        */
        if ( portal->verticalScroll == SCROLL_ON )
        {
            /*
            ** Adjust the vertical thumb on the right border
            */
            if ( portal->cursorLine == 0 )
            {
                if ( virtualHeight <= portalHeight )
                    curPos = 100;
                else
                    curPos = 0;
            }
            else
            {
                if ( portal->cursorLine >= bottomLine )
                    curPos = 100;
                else
                {
                    vLine = bottomLine;
                    if ( vLine == 0 )
                        vLine = 1;

                    curPos = ( portal->cursorLine *
                                100 ) / vLine;
                }
            }
            /*
            ** Erase the vertical thumb at its last position
            */
            portal->showScrollBars &= ~VERTICAL_SCROLL_MASK;
            /*
            ** Display the vertical thumb at its new positio
            */

```

```

OLL_SHIFT;

portal->showScrollBars |= curPos << VERTICAL_SCR
)
NWSUpdatePortal( portal );
updatedDisplay = FALSE;
)
NMSGgetKey( &keyType, &ch, NUTHandle );
switch ( keyType )
{
    case K_UP:
        /*
        ** Scroll the portal up one line.
        */
        if ( portal->virtualLine > 0 )
        {
            /*
            ** We're not at the top, so we can scrol
            */
            portal->virtualLine--;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        }
        break;

    case K_DOWN:
        /*
        ** Scroll the portal down one line.
        */
        if ( portal->virtualLine < bottomLine )
        {
            /*
            ** We're not at the bottom, so we can sc
            */
            portal->virtualLine++;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        }
        break;

    case K_PUP:
        /*
        ** Move the portal up one page.
        */
        if ( portal->cursorLine > 0 )
        {
            LONG delta;
            /*
            ** We're not at the top, so we can move
            */

```

```

** to figure out how much we can move up
*/
if ( portal->cursorLine > portalHeight )
    delta = portalHeight;
else
    delta = portal->cursorLine;

portal->cursorLine -= delta;
if ( portal->cursorLine < portal->virtua
    portal->virtualline = portal->cu

    updatedDisplay = TRUE;
)
break;

case K_PDOWN:
/* Move the portal down one page.
*/
if ( portal->cursorLine < virtualHeight )
(
    LONG delta;
    LONG newCurrentLine;
/*
** We're not at the bottom, so we can mo
** we need to figure out how much we can
*/
delta = virtualHeight - portal->cursorLi
if ( delta > portalHeight )
    delta = portalHeight;

newCurrentLine = portal->cursorLine + de
delta = virtualHeight - portalHeight;
if ( newCurrentLine > delta )
    portal->virtualline = delta;
else
    portal->virtualline = newCurrent
portal->cursorLine = portal->virtualline
updatedDisplay = TRUE;
)
break;

case K_SUP:
/* <Ctrl-PgUp> takes us to the top of the portal
*/
portal->virtualline = 0;
portal->cursorLine = 0;
updatedDisplay = TRUE;
}

void UpdateStatsInformation(LONG portal)
{
    PCB
    struct DriverStatsStructure *stats;

```

```

break;

case K_SDOWN:
/*
** <Ctrl-PgDn> takes us to the bottom of the por
*/
portal->cursorLine = virtualHeight;
portal->virtualline = portal->cursorLine - porta
updatedDisplay = TRUE;
break;

case K_ESCAPE:
    escapeFlag = 1;
break;

)
DPCGetSignalStrength(void)
struct DriverStatsStructure *stats;
CustVars
*customPtr;
LONG signal;
int beamSize, beamPercent;
if (DPCGetMLIDStats(&stats))
    return(0);

customPtr = (CustVars *)(&stats->CustomVariableCount);
signal = customPtr->CustomVariable[0];

if (signal >= 200)
    signal = (2 * (signal - 200)) + 50;
else
    signal = 0;

beamSize = (int)((signal - 60L)/2L);
beamSize *= 5;
beamSize /= 4;

if (beamSize < 0)
{
    beamSize = 0;
}
else if (beamSize >= MAX_BEAM)
{
    beamSize = MAX_BEAM;
}

beamPercent = (100*beamSize) / MAX_BEAM;
return(beamPercent);
}

void UpdateStatsInformation(LONG portal)
{
    PCB
    struct DriverStatsStructure *stats;

```

```

int line, count;
LONG
int numGenerics;
BYTE
LONG
LONG
LONG
LONG
CustVars
BYTE

NWSGetPCB( &portalPtr, portal, NUTHandle );

if (DPCGetMLIDStats(&stats))
{
    AddKey( NUTHandle->screenID, ENTER_KEY, 0, 0, 0 );
    return;
}

/* do generic stuff */

line = GenericLineStart;
statsPtr = &( stats->NotSupportedMask );
numGenerics = stats->GenericVariableCount;
mask = *statsPtr++;

for ( count = 0; count < numGenerics; count++ )
{
    if ( mask & ( 0x80000000 >> ( count & 0x1f ) ) )
    {
        /* not supported */
        NWSprintf( string, MSG("%13.13s", 301), MSG("Not support
ed", 298) );
        NWSShowPortalline
        (
            line++,
            GblDataCol,
            string,
            STATS_DATA_WIDTH,
            portalPtr
        );
        statsPtr++;
    }
    else
    {
        if ( count == 20 )
        {
            BYTE tmpString[ 80 ];

            /* This is the operating time stamp, which needs
            ** output format.
            */
            ConvertTicksToSeconds( *statsPtr++, &seconds, &t
            enths );
            FormatElapsedTime( seconds, tenths, tmpString );
            NWSprintf( string, MSG("%13.13s", 302), tmpStrin
            g );
        }
        else
        {
            CreateStringWithCommas
            (
                *statsPtr++,
                string,
                MSG("%13lu", 303)
            );
            NWSShowPortalline
            (
                line++,
                GblDataCol,
                string,
                STATS_DATA_WIDTH,
                portalPtr
            );
        }
        /* do custom stuff */

        line += 2;
        customPtr = (CustVars *)(&stats->CustomVariableCount);
        ptr = (BYTE *)(&customPtr->CustomVariable[customPtr->CustomVariableCount]);

        ptr += sizeof(WORD);

        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
            CreateStringWithCommas
            (
                customPtr->CustomVariable[ count ],
                string,
                MSG("%13lu", 299)
            );
            NWSShowPortalline( line++, GblDataCol, string, STATS_DATA_WIDTH,
            portalPtr );
        }
        NWSUpdatePortal( portalPtr );
    }
}

void SignalMeter(void) {
    int exitNow = FALSE;
    LONG type;
    BYTE value;
    int signal = 0;
    int oldSignal = 0;
    int avgSqf = 0;
    int beamSize;
    int beamPercent;
    int block = FALSE;
    int rxFreq;
    struct DriverStatsStructure *stats;
    CustVars *customPtr;
    char freqStr[80];
    char aveStr[80];
    char signalStr[80];
    int sound_gap = 0;

    while(!exitNow) {
        if (DPCGetMLIDStats(&stats)) {
            type = signal = 0;
            rxFreq = 0;
        }
        else {
            customPtr = (CustVars *)(&stats->CustomVariableCount);
            type = signal = customPtr->CustomVariable[0];
        }
    }
}

```

```

    rxFreq = customPtr->CustomVariable[1];
}

NWSprintf(freqStr, MSG("    Frequency : %d", 345), rxFreq / 10);

if (signal >= 200)
    signal = (2 * (signal - 200)) + 60;
else
    signal = 0;

if (avgSqf == 0L)
    avgSqf = 100L * signal;
avgSqf = ((avgSqf * 19L) + (100L * (unsigned long) signal)) / 20L;

beamSize = (int)((avgSqf/100L - 60L)/2L);
beamSize *= 5;
beamSize /= 4;

if (beamSize < 0)
    beamSize = 0;
else if (beamSize >= MAX_BEAM)
    beamSize = MAX_BEAM - 1;

beamPercent = (100*beamSize) / MAX_BEAM;

if (oldSignal != signal) {
    if (signal < MIN_SQF_VAL)
        bLock = FALSE;
    else
        bLock = TRUE;
    oldSignal = signal;
}

NWSprintf(aveStr, MSG("    Average SQF : %d", 346),
    signal ? avgSqf / 100L : 0);

NWSprintf(signalStr, MSG("Signal Quality : %d (%d%%) %s", 347),
    signal,
    beamPercent,
    bLock ? MSG("Signal Locked)", 348) : MSG("Signal Not Locked)", 3
49));

DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 341),
    beamSize,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);

if (--sound_gap <= 0) {
    /* calculate frequency based on raw signal strength */
    signal = 200 + type;
    /* adjust for 8253-5 timer chip output */
    signal = 1193180 / signal;

    /* turn on sound */
    outp(67, 182);
    outp(66, signal % 256);
    outp(66, signal / 256);
    outp(97, inp(97) | 0x03);

    delay(300);

    /* turn off sound */
    outp(97, inp(97) & ~0x03);
}

sound_gap = (110 - beamPercent) / 17;

delay(150);

if (NWSKeyStatus(NUTHandle)) {
    NWSGetKey(&type, &value, NUTHandle);
    if ((type == K_ESCAPE) || (type == K_AF10))
    {
        exitNow = TRUE;
    }
}

/* Destroy the throttle portal */
DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 351),
    MAX_BEAM,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);
}

void GetRegionName(char *regionName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 635), MSG("r", 636));
    len = strlen(MSG("RegionName=", 637));
    *regionName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("RegionName=", 638), len)) ==
                0)
            {
                fclose(fp);
                src = &szTmp[len];
                dest = regionName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                    & *src != '\n')
                {
                    *dest = *src;
                    src++;
                    dest++;
                }
                *dest = 0;
                return;
            }
            fclose(fp);
        }
    }

    void GetCountry(char *countryName)
    {
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;

```

```

fp = fopen(MSG("country.ini", 639), MSG("r", 640));
len = strlen(MSG("Name=", 641));
*countryName = 0;
if (fp != NULL)
{
    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        if ( (strnicmp(szTmp, MSG("Name=", 642), len)) == 0)
        {
            fclose(fp);
            src = &szTmp[len];
            dest = countryName;
            while(*src != 0 && *src != ' ' && *src != '\r' &
                *src != '\n')
            {
                *dest = *src;
                src++;
                dest++;
            }
            *dest = 0;
            return;
        }
        fclose(fp);
    }
}

void GetDefaultService(char *serviceName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 626), MSG("r", 627));
    len = strlen(MSG("Service=", 628));
    *serviceName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("Service=", 185), len)) == 0)
            {
                fclose(fp);
                src = &szTmp[len];
                dest = serviceName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                    *src != '\n')
                {
                    *dest = *src;
                    src++;
                    dest++;
                }
                *dest = 0;
                return;
            }
            fclose(fp);
        }
    }

    typedef struct
    {
        char name[20];
        LONG longitude;
        LONG eastFlag;
        LONG frequency;
        LONG horzFlag;
        LONG cityLongDegrees;
        LONG cityLongMinutes;
        LONG cityLatDegrees;
        LONG cityLatMinutes;
        LONG cityEastFlag;
        LONG cityNorthFlag;
    } SatelliteInfo;

    typedef struct
    {
        float elevation;
        float trueAzimuth;
        float magAzimuth;
        float polarization;
    } DishInfo;

    void ParseSatelliteInfo(char *satName, int nameContainsInfo, SatelliteInfo *sat)
    {
        FILE *fp;
        char szTmp[128];
        char *fptr;
        char *name, *cptr;
        char *ascPtr;
        char ascBuf[10];

        if (nameContainsInfo == FALSE)
        {
            fp = fopen(MSG("country.ini", 644), MSG("r", 629));
            if (fp != NULL)
            {
                while ((fptr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != N
                    ULL)
                {
                    if ( (strnicmp(szTmp, satName, strlen(satName)))
                        == 0)
                    {
                        break;
                    }
                    if (fp)
                        fclose(fp);
                    if (!fptr)
                        return;
                    cptr = szTmp;
                }
                else
                {
                    cptr = satName;
                    name = sat->name;
                    while(*cptr != '=')
                        *name++ = *cptr++;
                    *name = 0;
                    cptr++;
                    while(*cptr == ' ')
                        cptr++;
                    ascPtr = ascBuf;
                    while(*cptr != ',')
                        *ascPtr++ = *cptr++;
                }
            }
        }
    }
}

```

```

cptr++;
*ascPtr = 0;
sat->longitude = atoi(ascBuf);

if (*cptr == 'e' || *cptr == 'E')
    sat->eastFlag = 1;
else
    sat->eastFlag = 0;

while(*cptr != ',')
    cptr++;

ascPtr = ascBuf;
while(*cptr != ',')
    *ascPtr++ = *cptr++;

cptr++;
*ascPtr = 0;
sat->frequency = atoi(ascBuf);

if (*cptr == 'h' || *cptr == 'H')
    sat->horzFlag = 1;
else
    sat->horzFlag = 0;

)

void GetDefaultSatellite(SatelliteInfo *sat)
(
    FILE *fp;
    char szTmp[128];
    LONG len;
    char *ccode;

    fp = fopen(MSG("country.ini", 632), MSG("r", 633));
    len = strlen(MSG("[Hughes Networks]", 630));
    if (fp != NULL)
    (
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        (
            if ( ( strlen(szTmp, MSG("[Hughes Networks]", 631), len
                )) == 0)
            (
                ccode = fgets(szTmp, sizeof(szTmp) - 1, fp);
                fclose(fp);
                if (ccode == NULL)
                    return;

                ParseSatelliteInfo(szTmp, TRUE, sat);
                return;
            )
            fclose(fp);
        )
    )

    int NewCalculationFlag = 0;

LONG
(
    FILE *fp;
    SatelliteInfo *sat;
    List *listPtr = NULL;
    LONG ccode;
    LONG rcode = K_SELECT;

```



```

    NewCalculationFlag = 1;
    rcode = K_ESCAPE;
}

ChangeSatExit:
    NWSDestroyList(NUTHandle);
    NWSPopList(NUTHandle);
    NWSSetListSortFunction(NUTHandle, oldSortFunction);
    return rcode;
}

LONG
ComputeHotSpot(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
    fp = fp;
    key = key;
    changed = changed;
    handle = handle;
    NewCalculationFlag = 1;
    return K_ESCAPE;
}

int
LongitudeHemisphereHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
PolarizationHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
GroundLatHemHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
GroundLongHemHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

float SGN(float num)
{
    if (num < 0)
        return -1;
    return 1;
}

void CalculatedDish(SatelliteInfo *sat, DishInfo *dish)
{
    float LW, LN, ZR, YR, XR, DIST, EL, PO, AZ, TRAZ, S, A;
    float SD, RD, RM, LD, LM;
    static const float M_PI = 3.14159;
    static const float RAD = 6378.388;
    static const float ALT = 42164.24;
    int count = 4;

    SD = p->dSatLong;
    RD = p->dRemLongDeg;
    RM = p->dRemLongMin;

```

```

    LD = p->dRemLatDeg;
    LM = p->dRemLatMin;

    SD = (float)sat->longitude;
    RD = (float)sat->cityLongDegrees;
    RM = (float)sat->cityLongMinutes;
    LD = (float)sat->cityLatDegrees;
    LM = (float)sat->cityLatMinutes;

    SD = fabs(SD);
    if (p->cSatLong == 'E')
        if (sat->eastFlag)
            SD = -SD;

    RD = fabs(RD);
    RM = fabs(RM);
    if (p->cRemLong == 'E')
        if (sat->cityEastFlag)
        {
            RD = -RD;
            RM = -RM;
        }

    LD = fabs(LD);
    LM = fabs(LM);
    if (p->cRemHem == 'S')
        if (sat->cityNorthFlag == 0)
        {
            LD = -LD;
            LM = -LM;
        }
    LW = ((RD + RM / 60) - SD) * M_PI / 180;

    if (LW == 0)
        LW = .00001;
    LN = (LD + LM / 60) * M_PI / 180;
    if (LN == 0)
        LN = .00001;

    ZR = RAD * sin(LN);
    YR = RAD * cos(LN) * cos(LW);
    XR = RAD * cos(LN) * sin(LW);
    DIST = sqrt(XR * XR + (ALT - YR) * (ALT - YR) + ZR * ZR);
    EL = (ALT * YR - RAD * RAD) / (RAD * DIST);
    EL = atan(EL / sqrt(1 - EL * EL)) * 180 / M_PI;
    EL = (10 * EL + .5 * SGN(EL)) / 10;
    A = 0;

    if (LN * LW > 0)
        A = 2 * M_PI;
    if (LN > 0)
        A = M_PI;

    AZ = (A - atan(tan(LW) / sin(LN))) * 180 / M_PI;
    AZ = floor(10 * AZ + .5 * SGN(AZ)) / 10;
    TRAZ = AZ;

    /* Executed for US Mainland only */
    /* Declination correction to TRUE Azimuth */
    if (((LD > 23) && (LD < 50)) && ((RD > 70) && (RD < 125)))

```

```

(
    LD = floor(LD/4) - 6;
    RD = ceil(RD/4) - 18;
    if (inDeclination((int) (LD*14+RD)))
        count--;
    if (inDeclination((int) (LD*14+RD-1)))
        count--;
    if (inDeclination((int) ((LD+1)*14+RD-1)))
        count--;
    if (inDeclination((int) ((LD+1)*14+RD)))
        count--;
    if (count)
    {
        AZ = AZ + nDeclination((int) (LD*14+RD))
        + nDeclination((int) (LD*14+RD-1))
        + nDeclination((int) ((LD+1)*14+RD))
        + nDeclination((int) ((LD+1)*14+RD-1))/count;
    }

    if (AZ >= 360) AZ = AZ - 360;
    if (AZ < 0) AZ = AZ + 360;

    S = tan(LN) / sin(LW);

    PO = atan(S);
    /* printf("S=%f LN=%f LW=%f PO=%f\n", S, LN, LW, PO); */

    PO = fabs(PO);
    PO = M_PI / 2.0 - PO;
    PO = PO * SGN(S) * 180 / M_PI;
    PO = floor(10 * PO + .5 * SGN(PO)) / 10;

    //
    //
    //
    //
    p->dRemElev = EL;
    p->dRemMagAz = AZ;
    p->dRemTrueAz = TRAZ;
    p->dRemPolar = -PO;
    dish->elevation = EL;
    dish->magAzimuth = AZ;
    dish->trueAzimuth = TRAZ;
    dish->polarization = -PO;

    )

void ComputeCity(char *region, char *city, LONG latLong)
{
    FIELD *fp;
    char country[20], countryStr[30];
    char regionStr[30], cityStr[30];
    char service[30], serviceStr[30];
    char satelliteStr[50];
    char elevationStr[50], trueAzimuthStr[50];
    char magAzimuthStr[50], polarizationStr[50];
    SatelliteInfo sat;
    DishInfo dish;
    int i;
    int start;
    MFCNTROL *mfctl0, *mfctl1, *mfctl2, *mfctl3;

    calculateAgain:
    NWSInitForm(NUTHandle);
    if (NewCalculationFlag == 0)
    {

```

```

        sat.cityLatDegrees = (latLong >> 24) & 0xff;
        sat.cityLatMinutes = (latLong >> 16) & 0xff;
        sat.cityLongDegrees = (latLong >> 8) & 0xff;
        sat.cityLongMinutes = (latLong & 0xff);
        GetDefaultSatellite(&sat);
        sat.cityEastFlag = sat.eastFlag;
        sat.cityNorthFlag = 1;
    }
    NewCalculationFlag = 0;

    i = 0;
    GetCountry(country);
    NWSprintf(countryStr, MSG("Country : %s", 488), country);
    NWSAppendCommentField(i, 2, countryStr, NUTHandle);

    NWSprintf(regionStr, MSG("Region : %s", 712), region);
    NWSAppendCommentField(i, 36, regionStr, NUTHandle);

    i++;
    NWSprintf(cityStr, MSG("City : %s", 713), city);
    NWSAppendCommentField(i, 2, cityStr, NUTHandle);

    GetDefaultService(service);
    NWSprintf(serviceStr, MSG("Service : %s", 714), service);
    NWSAppendCommentField(i, 36, serviceStr, NUTHandle);

    i++;
    NWSprintf(satelliteStr, MSG("Satellite : %s", 649), sat.name);
    start = (78 - strlen(satelliteStr)) / 2;
    fp = NWSAppendHotSpotField(i, start, NORMAL_FIELD, satelliteStr,
        ChangeSatellite, NUTHandle);
    fp->customData = &sat;

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Longitude : ", 715), NUTHa
        ndle);
    NWSAppendIntegerField(i, 27, NORMAL_FIELD, (int *)&sat.longitude, 1, 180
        , F_NO_HELP, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Hemisphere : ", 716), NUTHandle);
    mfctl0 = NWSInitMenuField(InxMSG("Satellite Longitude Hemisphere", 717),
    10, 40, LongitudeHemisphereHandler, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("West", 718), 0, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("East", 719), 1, NUTHandle);
    NWSAppendMenuField(i, 47, NORMAL_FIELD, (int *)&sat.eastFlag, mfctl0, NU
        LL, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Polarization : ", 489), NUTHa
        ndle);
    mfctl1 = NWSInitMenuField(InxMSG("Satellite Polarization", 490), 10, 40,
    PolarizationHandler, NUTHandle);
    NWSAppendToMenuField(mfctl1, InxMSG("Vert", 531), 0, NUTHandle);
    NWSAppendToMenuField(mfctl1, InxMSG("Horz", 539), 1, NUTHandle);
    NWSAppendMenuField(i, 27, NORMAL_FIELD, (int *)&sat.horzFlag, mfctl1, NU
        LL, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Frequency : ", 541), NUTHandle);
    NWSAppendIntegerField(i, 47, NORMAL_FIELD, (int *)&sat.frequency, 1, 100
        00, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Ground Longitude degrees : ", 543), NUT
        Handle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLongDegrees,
    1, 180, F_NO_HELP, NUTHandle);

```

```

NWSAppendCommentField(i, 40, MSG("minutes : ", 545), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLongMinutes,
1, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 686), NUTHandle);
mfc12 = NWSInitMenuField(InxMSG("Ground Longitude Hemisphere", 687), 10
, 40, GroundLongHemHandler, NUTHandle);
NWSAppendToMenuField(mfc12, InxMSG("West", 688), 0, NUTHandle);
NWSAppendToMenuField(mfc12, InxMSG("East", 689), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityEastFlag, mfc12
, NULL, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("Ground Latitude degrees : ", 690), NUT
Handle);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLatDegrees, 1
, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 40, MSG("minutes : ", 691), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLatMinutes, 1
, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 692), NUTHandle);
mfc13 = NWSInitMenuField(InxMSG("Ground Latitude Hemisphere", 693), 10,
40, GroundLatHemHandler, NUTHandle);
NWSAppendToMenuField(mfc13, InxMSG("South", 694), 0, NUTHandle);
NWSAppendToMenuField(mfc13, InxMSG("North", 695), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityNorthFlag, mfc1
3, NULL, NUTHandle);

i++;
NWSAppendHotSpotField(i, 35, NORMAL_FIELD, MSG("COMPUTE NOW", 696),
ComputeHotSpot, NUTHandle);

CalculatedDish(&sat, &dish);

if (sat.horzFlag == 0)
{
    dish.polarization += 90.0;
    if (dish.polarization > 90.0)
        dish.polarization -= 90.0;
    if (dish.polarization < -90.0)
        dish.polarization += 90.0;
}

i+=2;
NWSprintf(elevationStr, MSG(" Elevation : %lf", 697), dish.elev
ation);
NWSprintf(trueAzimuthStr, MSG(" True Azimuth : %lf", 698), dish.true
Azimuth);
// if ((strcmp(country, MSG("USA", 243))) != 0)
//     strcpy(magAzimuthStr, MSG("Magnetic Azimuth : N/A", 699));
// else
//     NWSprintf(magAzimuthStr, MSG("Magnetic Azimuth : %lf", 568), d
ish.magAzimuth);
NWSprintf(polarizationStr, MSG(" Polarization : %lf", 700), dish.pola
rization);

NWSAppendCommentField(i, 2, elevationStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, trueAzimuthStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, magAzimuthStr, NUTHandle);
i++;

```

```

NWSAppendCommentField(i, 2, polarizationStr, NUTHandle);
i++;

```

```

NWSeditPortalForm(InxMSG("Antenna Pointing Calculations", 701),
12, 40,
/* center line, column */
i, 78,
/* form height, width */
F_NOVERIFY, F_NO_HELP,
/* Control flags, help m
essage */
NULL,
NUTHandle);
/* Confirm message, hand
le */

```

```

NWSDestroyForm(NUTHandle);
if (NewCalculationFlag != 0)
    goto calculateAgain;
}

```

```

void ComputeGetCity(char *region)
{
    DIR *dirCountry, *dirCity;
    char *name;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;
    FILE *fp;
    char szTmp[128];
    char byteStr[8];
    int latDeg, latMin, longDeg, longMin;
    int len, start = 0;
    LONG info;
}

```

```

NWSStartWait( 0, 0, NUTHandle ); /* DMH change 970131 */
getCitiesLoop:
    rows = cols = 0;
    listPtr = NULL;
    NWSInitList(NUTHandle, NULL);

```

```

    dirCountry = opendir(MSG("*.cty", 702));
    if (dirCountry == NULL)
        goto errorNoDir;

```

```

    while( (dirCity = readdir(dirCountry)) != NULL )
    {

```

```

        name = dirCity->d_name;

```

```

        fp = fopen(name, MSG("r", 703));
        if (fp == NULL)
        {

```

```

            closedir(dirCountry);
            goto errorNoDir;
        }

```

```

        len = strlen(region);
        if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(region, szTmp, len)) == 0)
                break;
        }
        fclose(fp);
    }
}

```

```

if (dirCity == NULL)
{
    closedir(dirCountry);
    goto errorNoDir;
}

while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
{
    int len = strlen(szTmp) - 2;
    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[3] = 0;
    longMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[0] = szTmp[len--];
    longDeg = atoi(&byteStr);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latDeg = atoi(&byteStr[1]);

    if (longMin > 59)
    {
        longDeg++;
        longMin = 0;
    }

    if (latMin > 59)
    {
        latDeg++;
        latMin = 0;
    }

    info = ((latDeg & 0xff) << 24) |
            ((latMin & 0xff) << 16) |
            ((longDeg & 0xff) << 8) |
            (longMin & 0xff);

    while(szTmp[len] == ' ')
        len--;
    if (len > 0)
    {
        szTmp[len+1] = 0;
        while(len)
        {
            if (szTmp[len] == ' ')
                start = len + 1;
            len--;
        }
        if (start > 0)
        {
            AppendToList(&szTmp[start], info, &rows, &cols);
        }
    }
    fclose(fp);
    if (rows == 0)
    {

```

```

        closedir(dirCountry);
        goto errorNoDir;

        NWSWait( NUTHandle ); /* DWH change 970131 */
        ccode = NWSList(
            12, 40,
            (rows < 16) ? rows : 16,
            (cols < 18) ? 18 : cols,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        if (ccode == M_SELECT)
        {
            info = (LONG)listPtr->otherInfo;
            strcpy(szTmp, listPtr->text);
            NWSDestroyList(NUTHandle);
            closedir(dirCountry);
            ComputeCity(region, szTmp, info);
            goto getCitiesLoop;
        }

        NWSDestroyList(NUTHandle);
        closedir(dirCountry);
        return;

    errorNoDir:
        NWSWait( NUTHandle ); /* DWH change 970131 */
        NWSDestroyList(NUTHandle);
        NWSAlert(12, 40, NUTHandle, InxMSG("Unable to access Cities file", 705))
        ;
        return;
    }

    void ComputeGetRegion(char *country)
    {
        DIR *dirCountry, *dirCity;
        char *name, *end;
        LIST *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;
        char path[64];
        FILE *fp;
        char szTmp[128];
        LONG header;

        NWSWait( 0, 0, NUTHandle ); /* DWH change 970131 */
        getRegionsLoop:
            rows = cols = 0;
            listPtr = NULL;
            NWSInitList(NUTHandle, NULL);
            SetCurrentNameSpace(DOSNameSpace);
            NWSprintf(path, MSG("SYS:DIREPCP\\DB\\%s.cou", 706), country);
            if (chdir(path))
                goto errorNoDir;

            dirCountry = opendir(MSG("*.cty", 707));
            if (dirCountry == NULL)
                goto errorNoDir;

```

```

while( (dirCity = readdir(dirCountry)) != NULL )
{
    name = dirCity->d_name;
    fp = fopen(name, MSG("r", 708));
    if (fp == NULL)
    {
        closedir(dirCountry);
        goto errorNoDir;
    }
    if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        end = &szTmp[strlen(szTmp) - 2];
        while( (*end == ' ') && (end != szTmp) )
            end--;
        end++;
        *end = 0;
        AppendToList(szTmp, 0, &rows, &cols);
    }
    fclose(fp);
}

if (rows == 0)
{
    closedir(dirCountry);
    goto errorNoDir;
}

GetRegionName(szTmp);
if ( (strcmpi(szTmp, MSG("Province", 709))) == 0)
    header = InxMSG("Choose A Province", 710);
else
    header = InxMSG("Choose A State", 711);

if (rows == 1)
{
    NWSEndWait( NUTHandle );
    NWSDestroyList(NUTHandle);
    ComputeGetCity(szTmp);
    closedir(dirCountry);
    return;
}
else
{
    NWSEndWait( NUTHandle );
    ccode = NWSList(
        header,
        12, 40,
        (rows < 16) ? rows : 16,
        (cols < 18) ? 18 : cols,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

    /* Height */
    /* Width */

    if (ccode == M_SELECT)
    {
        strcpy(szTmp, listPtr->text);
        NWSDestroyList(NUTHandle);
        closedir(dirCountry);
        ComputeGetCity(szTmp);
        goto getRegionsLoop;
    }
}

```

```

NWSDestroyList(NUTHandle);

```

```

closedir(dirCountry);
return;

```

```

errorNoDir:
    NWSEndWait( NUTHandle );
    NWSDestroyList(NUTHandle);
    NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country files in Co
untry directory %s.cou", 569), country);
    return;
}

```

```

void ComputeCoordinates(void)
{

```

```

    DIR *dirDB, *dirCountry;
    char *name, *dot;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;

```

```

getCountriesLoop:
    rows = cols = 0;
    listPtr = NULL;
    NWSInitList(NUTHandle, NULL);
    SetCurrentNameSpace(DOSNameSpace);
    if (chdir(MSG("SYS:DIRECPC\\DB", 570)))
        goto errorNoDir;

```

```

    dirDB = opendir(MSG("*.cou", 571));
    if (dirDB == NULL)
        goto errorNoDir;

```

```

    while( (dirCountry = readdir(dirDB)) != NULL )
    {
        name = dirCountry->d_name;
        if ((dot = strchr(name, '.')) != NULL)
            *dot = 0;
        AppendToList(name, 0, &rows, &cols);
    }

```

```

    if (rows == 0)
    {

```

```

        closedir(dirDB);
        goto errorNoDir;
    }

```

```

    ccode = NWSList(
        InxMSG("Choose A Country", 572),
        12, 40,
        rows,
        18,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

```

```

    /* Height */
    /* Width */

```

```

    if (ccode == M_SELECT)
    {
        if (NWSPushList(NUTHandle) != 0)
        {
            name = listPtr->text;

```

```

        ComputeGetRegion(name);
        NWSPopList (NUTHandle);
    }
    NWSDestroyList (NUTHandle);
    closedir (dirDB);
    goto getcountriesLoop;
}

NWSDestroyList (NUTHandle);

closedir (dirDB);
return;

errorNoDir:
    NWSDestroyList (NUTHandle);
    NWSAlert (12, 40, NUTHandle, InxMSG("Unable to locate Country directories
    in SYS:DIRECPC\\DB", 573));
    return;
}

void DPCPointing(void)
{
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;

    NWSInitList (NUTHandle, NULL);

    NWSAppendToList (MSG("Antenna Pointing Calculations", 574), (void *)1, NUT
    Handle);
    NWSAppendToList (MSG("Signal Strength Meter", 575), (void *)2, NUTHandle)
    ;
    while (ccode != M_ESCAPE)
    {
        ccode = NWSList(
            InxMSG("Dish Pointing", 576),
            12, 40,
            2,
            30,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        /* Heigh
        /* Width

t */
        /*

        if (ccode == M_SELECT)
        {
            if (NWSPushList(NUTHandle) != 0)
            {
                switch ((int)listPtr->otherInfo)
                {
                    case 1:
                        ComputeCoordinates();
                        break;

                    case 2:
                        SignalMeter();
                        break;

                    default:
                        break;
                }
            }
            NWSPopList (NUTHandle);
        }
    }
}
    dpPtr = (MUXdcau_t *)CASDBdcau.p_buffer;
    while (count)
    {
        line++;
        if (line > (portalPtr->virtualHeight - 1))
            break;
    }
}

NWSDestroyList(NUTHandle);

void UpdateAdapterInformation(LONG portal)
{
    PCB
    int *portalPtr;
    MUXdcau_t *dpPtr;
    BYTE string[80];
    char serialNumber[10];

    NWSGetPCB (&portalPtr, portal, NUTHandle );

    /* do generic stuff */

    line = 0;

    DIOGetSN(serialNumber);
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        serialNumber,
        CStrLen(serialNumber),
        portalPtr
    );

    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        SiteID,
        CStrLen(SiteID),
        portalPtr
    );

    NWSprintf(string, MSG("%s", 503), CASDBdcau.entries == 0 ? MSG("FALSE",
    474) : "TRUE ");
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    count = CASDBdcau.entries;
    NWSprintf( string, MSG("%d", 495), count );
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    dpPtr = (MUXdcau_t *)CASDBdcau.p_buffer;
    while (count)
    {
        line++;
        if (line > (portalPtr->virtualHeight - 1))
            break;
    }
}

```

```

for (col = 8; col <= 64; col += 16, count--)
{
    if (!count)
        break;

    NWSprintf( string, MSG("%2.2X%2.2X%2.2X%2.2X", 504),
        dptr->groupid.i[2], dptr->groupid.i[1],
        dptr->groupid.i[0], dptr->version);

    NWSShowPortallLine(
        line,
        col,
        string,
        CStrLen(string),
        portalPtr
    );

    dptr++;
}

NWSUpdatePortal( portalPtr );

}

void DisplayAdapterInfo(void)
{
    int line, len;
    BYTE oldPortal;
    PCB *portalPtr;
    BYTE string[80];

    line = 5 + 3; /* Site ID, S/N, Key Status, Number of Communities,
        Current Communities: */
    extra community lines */
    line += (CASDBdcau.entries / 4);

    oldPortal = NUTHandle->currentPortal;
    NWSDeSelectPortal( NUTHandle );
    BackgroundPortal = NWSCreatePortal
    (
        1 /* gblUPFTopLine */,
        1 /* gblUPFLeftCol */,
        ((line + 4) > (ScreenHeight - 3)) ? ScreenHeight - 3 : line + 4,
        /* gblUPFHeight + gblUPFHeight */
        ScreenWidth - 2 /* gblUPFWidth */,
        line,
        ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
        TRUE,
        MSG("DPC Adapter Information", 502),
        VNORMAL,
        SINGLE,
        VINTENSE,
        CURSOR_OFF,
        VIRTUAL,
        NUTHandle
    );

    if ( BackgroundPortal > MAXPORTALS )
    {
        NWSDeSelectPortal( oldPortal, NUTHandle );
        return;
    }

    NWSGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
    portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
    portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
    portalPtr->verticalScroll = SCROLL_ON;

```

```

NWSClearPortal( portalPtr );

```

```

/* Start filling in the static portal lines.
*/

```

```

line = 0;
NWSprintf(string, MSG("Serial Number
len = CStrLen( string );
NWSShowPortallLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Site ID
len = CStrLen( string );
NWSShowPortallLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Have Keys Status
len = CStrLen( string );
NWSShowPortallLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Number Of Communities
len = CStrLen( string );
NWSShowPortallLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Communities
len = CStrLen( string );
NWSShowPortallLine( line++, BORDER_WIDTH, string, len, portalPtr );

GblDataCol = BORDER_WIDTH + 24;

UpdateAdapterInformation(BackgroundPortal);
BackgroundFuncPtr = UpdateAdapterInformation;
HandleScrollablePortal(portalPtr);
BackgroundFuncPtr = NULL;
NWSDeSelectPortal( BackgroundPortal, NUTHandle );
NWSDeSelectPortal( oldPortal, NUTHandle );

```

```

void DisplayMLIDStats(void)
{

```

```

    int line;
    int count;
    int numberOfGenerics;
    int len;
    int promptMax;

    struct DriverStatsStructure *stats;
    struct DriverConfigurationStructure *config;
    ProtocolNodeStructure *protocol;
    CustomVars *customPtr;
    BYTE *customStrings, *ptr;
    BYTE oldPortal;
    BYTE name[128], *namePtr;
    PCB *portalPtr;
    BYTE string[80];

```

```

GblDataCol = ( ScreenWidth - 4 ) - BORDER_WIDTH - STATS_DATA_WIDTH;
if (DPCGetMLIDStats(&stats))
{
    return;
}

```

```

DIOGetMLIDConfig(&config);

```

```

/* Build up the LAN name followed by its I/O resources

```

```

* to be used as the portals header.
*/

```

```

if (config->DLogicalName[0] != 0)
    NWSprintf(name, MSG("%s [%s", 270), config->DLogicalName, config
->DShortName);
else
    NWSprintf(name, MSG("%s", 271), config->DShortName);

if (config->DIOPortsAndRanges[1] > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" port=%X", 272), config->DIOPortsAndRang
es[0]);
}
if (config->DIOPortsAndRanges[3] > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" port=%X", 273), config->DIOPortsAndRang
es[2]);
}
if (config->DMemoryDecodeAndLength[0].LANMemoryAddress > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 274),
        config->DMemoryDecodeAndLength[0].LANMemoryAddre
ss);
}
if (config->DMemoryDecodeAndLength[1].LANMemoryAddress > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 275),
        config->DMemoryDecodeAndLength[1].LANMemoryAddre
ss);
}
if (config->DIntLine[0] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" int=%X", 276), config->DIntLine[0]);
}
if (config->DIntLine[1] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" int=%X", 277), config->DIntLine[1]);
}
if (config->DDMALine[0] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 278), config->DDMALine[0]);
}
if (config->DDMALine[1] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 279), config->DDMALine[1]);
}
if (config->DMediaType != NULL)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" frame=%s", 280), config->DMediaType);
}
namePtr = name + CStrLen(name);
NWSprintf(namePtr, MSG(" ", 281));

```

```

/*
* Before we can create the portal, we must add up the number
* of lines we'll need, which will be bigger than the window.

```

```

line = 3;
protocol header */

/* Add up the number of protocols bound to this board */
protocol = MLIDProtocolListByBoard( DIOBoard );
while (protocol != NULL)
{
    ++line;
    protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
}

/* Add in the number of generic statistics */
line += 2; /* Blank line and Generic statistics header */

numberOfGenerics = stats->GenericVariableCount;
line += numberOfGenerics;

promptMax = 0;
for (count = 0; count < numberOfGenerics ; count++)
{
    len = CStrLen( GetMsg(GenericDescriptionTable(count)) );
    if (promptMax < len)
    {
        promptMax = len;
    }
}

/* Add in the number of custom statistics */
line += 2; /* Blank line and Custom statistics header */

customPtr = (CustVars *)(&stats->CustomVariableCount);
customStrings = (BYTE *) (customPtr->CustomVariable[customPtr->CustomVari
ableCount]);
customStrings += sizeof(WORD);

line += customPtr->CustomVariableCount;

/* Check lengths of custom strings */

ptr = customStrings; /* temp pointer to walk down custom prompts */
for ( count = 0; count < customPtr->CustomVariableCount; count++)
{
    len = CStrLen( ptr );
    if ( promptMax < len )
        promptMax = len;
    while ( *( ptr++) )
        ; /* find the next string */
}

line += 1; /* add a blank line for the bottom */

if ( promptMax < 46 )
    promptMax = 64; /* minimum portal width */
else if ( promptMax > 60 )
    promptMax = 76; /* maximum portal width */
else
    promptMax += 16; /* custom portal width within range */

/* build the portal */

```



```

if ( line < 8 )
    line = 8;
/* must meet the minimum portal size */

oldPortal = NUTHandle->currentPortal;
NWSeselectPortal( NUTHandle );
BackgroundPortal = NWSCreatePortal
(
    1 /* gblUPFTopLine */,
    1 /* gblUPFLeftCol */,
    ScreenHeight - 3 /* gblUPFHeight + gblUPFHeight */,
    ScreenWidth - 2 /* gblUPFWidth */,
    line,
    ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
    TRUE,
    name,
    VNORMAL,
    SINGLE,
    VINTENSE,
    CURSOR_OFF,
    VIRTUAL,
    NUTHandle
);
if ( BackgroundPortal > MAXPORTALS )
{
    NWSSelectPortal( oldPortal, NUTHandle );
    return;
}

NWSgetPCB( &portalPtr, BackgroundPortal, NUTHandle );
portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
portalPtr->verticalScroll = SCROLL_ON;
NWSclearPortal( portalPtr );

/* Start filling in the static portal lines.
*/

/* Fill in the MLID version */
line = 0;
NWSPrintf( string, MSG("Version %d.%d", 282), config->DMLID_MajorVersion,
config->DMLID_MinorVersion);
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* Fill in the node address */
NWSPrintf( string, MSG("Node Address: %x%4.4x", 283),
GET_HILO_LONG( (LONG *) &config->DNodeAddress[0] ),
GET_HILO_WORD( (WORD *) &config->DNodeAddress[4] ));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* protocols */
NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Protocols:", 284),
CStrLen( MSG("Protocols:", 287) ), portalPtr );
protocol = MLDProtocolListByBoard( DIOBoard );
while ( protocol != NULL )
{
    GetProtocolNameTableEntry( protocol->ProtocolNumber, string );
    CMovB( ProtocolNameTable[protocol->ProtocolNumber], string, 16 );
    len = string[ 0 ];
    string[ len + 1 ] = NULL;
    NWSShowPortalLine

```

```

(
    line++,
    BORDER_WIDTH + INDENT_WIDTH,
    &string[ 1 ],
    len,
    portalPtr
);
if ( LStrCmp( string, MSG("\x003IPX", 285) ) == 0 )
{
    LONG tmpLong;
    /* network address */

    tmpLong = IPXNetNumberTable[ DIOBoard ];
    NWSPrintf( string, MSG("Network address: ", 286), GET_H
ILO_LONG( &tmpLong ));
    len = CStrLen( string );
    NWSShowPortalLine
    (
        line++,
        BORDER_WIDTH + ( 2 * INDENT_WIDTH ),
        string,
        len,
        portalPtr
    );
    protocol = (ProtocolNodeStructure *) protocol->ProtocolBoardLink;
}
/* Generic Statistics */
line++;
len = CStrLen( MSG("Generic statistics", 288) );
NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Generic statistics", 289),
len, portalPtr );
GenericLineStart = line;
promptMax = 16;

for ( count = 0; count < numberOfGenerics; count++ )
{
    ptr = GetMsg( GenericDescriptionTable[ count ] );
    len = CStrLen( ptr );
    if ( len > promptMax )
        len = promptMax;
    NWSShowPortalLine
    (
        line++,
        BORDER_WIDTH + INDENT_WIDTH,
        ptr,
        len,
        portalPtr
    );
}
/* custom stats */
line++;
len = CStrLen( MSG("Custom statistics", 227) );
NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Custom statistics", 292),
len, portalPtr );

for ( count = 0; count < customPtr->CustomVariableCount; count++ )
{
    NWSShowPortalLine
    (
        line++,

```

```

        BORDER_WIDTH + INDENT_WIDTH,
        customStrings,
        CStrLen( customStrings ),
        portalPtr
    );
    while ( *( customStrings++ ) )
        /* find the next string */
    }

```

```

    UpdateStatsInformation(BackgroundPortal);
    BackgroundFuncPtr = UpdateStatsInformation;
    HandleScrollablePortal(portalPtr);
    BackgroundFuncPtr = NULL;
    NWSDestroyPortal( BackgroundPortal, NUTHandle );
    NWSselectPortal( oldPortal, NUTHandle );

```

```

    LONG DisconnectRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
    {

```

```

        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

```

```

        DloEndConn();

```

```

        return(K_NORMAL);
    }

```

```

    LONG DialInternetRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
    {

```

```

        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

```

```

        DloStartConn(DLO_INET_TIMEOUT);

```

```

        return(K_NORMAL);
    }

```

```

    LONG DialPDRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
    {

```

```

        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

```

```

        DloStartConn(DLO_PACKAGE_TIMEOUT);

```

```

        return(K_NORMAL);
    }

```

```

    LONG SendModemRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
    {

```

```

        BYTE sendStr[82];
        int ccode;
        LONG len;
        fp = fp;
        key = key;

```

```

        changed = changed;
        handle = handle;

```

```

        sendStr[0] = 0;

```

```

        if (!NWSPushList(NUTHandle))
            return(K_NORMAL);

```

```

        ccode = NWSeditString(
            12, 40,
            /* center line, column */
            1, 40,
            /* edit height, width */

```

```

            /* header
            InxMSG("Modem Send Editor", 587),

```

```

            /* prompt
            InxMSG("Send : ", 588),

```

```

            /* buffer, max len
            (BYTE*)&sendStr, 80,

```

```

            /* type, handle
            EF_ANY, NUTHandle,
            NULL, NULL,

```

```

            /* insert Proc, action Proc*/
            MSG("a..z0..9A..Z-_", 589));
            if ( ccode & E_ESCAPE )
            {

```

```

                /* Start change by DWH 961115 */
                NWSPopList(NUTHandle);
                /* End change by DWH 961115 */
                return(K_NORMAL);
            }

```

```

            if ( (len = CStrLen(sendStr)) )
            {
                DloSend(sendStr, len, DLO_INET_TIMEOUT);
                DloSend(MSG("\r", 609), 1, DLO_INET_TIMEOUT);

```

```

                NWSPopList(NUTHandle);
                return(K_NORMAL);
            }

```

```

        void ModemControl(void)
        {

```

```

            int i;

```

```

            NWSInitForm(NUTHandle);

```

```

            i = 0;

```

```

            NWSAppendHotSpotField(i, 15-(20/2), NORMAL_FIELD, MSG("Disconnect the Mo
            dem", 547),

```

```

            i++;
            DisconnectRoutine, NUTHandle);

```

```

            NWSAppendHotSpotField(i, 15-(18/2), NORMAL_FIELD, MSG("Dial the Internet
            ", 549),

```

```

            DialInternetRoutine, NUTHandle);

```

```

            i++;
            NWSAppendHotSpotField(i, 15-(26/2), NORMAL_FIELD, MSG("Dial the Package
            Delivery", 590),

```

```

            DialPDRoutine, NUTHandle);

```

```

            i++;
            NWSAppendHotSpotField(i, 15-(18/2), NORMAL_FIELD, MSG("Send to the Modem

```

```

", 551),
    i++;
    NWSeditPortalForm(InxMSG("Modem Control Options", 552),
        10, 30, column, * /
        i, 30,
        /* form height, width */
        /* F_NOVERIFY, F_NO_HELP,
        /* Control flags, help m
        message */
        InxMSG("Save Changes?", 585),
        NUTHandle);
    /* Confirm message, hand
    le */

    NWSDestroyForm(NUTHandle);
}
/*****
MainOptionsHandler(void)
Description:
    This routine is where the main agent thread lives.
    It initiates the NUT screen, waits for the DPC MLID
    to become active, and wait for user commands.
Input:    nothing
Output:   nothing
Returns:  nothing
*****/
void MainOptionsHandler()
{
    int choice, prevChoice, i;
    int exitFlag = FALSE;
    LIST *defaultList;
    LONG type;
    BYTE value;
    int countdown = MAX_COUNTDOWN;
    LONG mainPortal;
    LONG removedCount;

    #if DRIVER_IO
    void (*ControlEntryPoint) () = NULL;
    LONG board;
    struct DriverConfigurationStructure *config;
    char *configName;
    #endif

    /*
    * Clear the screen by creating huge portal with VNORMAL attribute.
    */

    GetScreenSize(&ScreenHeight, &ScreenWidth);
    ScreenHeight = ScreenHeight - NUTHandle->headerHeight;
    mainPortal = NWSCreatePortal(
        NUTHandle->headerHeight, 0,
        /* line, column
        */
        SendModemRoutine, NUTHandle);
    width
    t/width
    /* Save flag, header text
    r attr, border type
    r attr, cursor flag
    t flag, handle
    if (mainPortal >= MAXPORTALS)
        return;

    #if DRIVER_IO
    LookForAdapter:
    #endif

    while (NWSKeyStatus(NUTHandle))
        NWSGetKey(&type, &value, NUTHandle);

    NWSUpdatePortal( NUTHandle->portal[mainPortal] );

    /*
    * Display our server name in an info portal at the top of the screen.
    */

    UpdateHelpPortal();

    /*
    * Lets look for a DPC adapter.
    */

    StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle, InxMSG("
    Waiting for DPC adapter to load", 143));
    DPCName[0] = 3;
    #if DRIVER_IO

    while(DIORegisterWithAdapter(DPCName) && exitFlag == FALSE)
    {
        delay(500);
        if (NWSKeyStatus(NUTHandle))
        {
            NWSGetKey(&type, &value, NUTHandle);
            if ((type == K_ESCAPE) || (type == K_AF10))
            {
                NWSEndWait(NUTHandle);
                return;
            }
        }
        if (--countdown == 0)
        {
            Spin(NUTHandle);
            countdown = MAX_COUNTDOWN;
        }
    }
    if (exitFlag)
        return;
    removedCount = DIORemovedCount;
    choice = prevChoice = PackageDelivery ? 1 : 2;
    while(1)
    {
        for(board = 0; board < NumberOfLANs; board++)

```

```

    oint);
    CLSLGetMLIDControlEntry(board, (void(*)())&ControlEntryP
    if (ControlEntryPoint)
    {
        config = (struct DriverConfigurationStructure *)
            CommandMlid(board
d, 0, (LONG)ControlEntryPoint);
        if (config)
        {
            configName = config->DShortName;
            if (!CStrCmp(configName, DPCName))
                goto FoundDPCBoardNumber;
        }
    }

    delay(500);
    if (NWSKeyStatus(NUTHandle))
    {
        NWSGetKey(&type, &value, NUTHandle);
        if ((type == K_ESCAPE) || (type == K_AF10))
        {
            NWSEndWait(NUTHandle);
            return;
        }
    }
    if (--countdown == 0)
    {
        Spin(NUTHandle);
        countdown = MAX_COUNTDOWN;
    }
}

/*
 * OK. We have a DPC MLID. Lets set up the main menu and
 * wait for the user to do something.
 */

#ifdef DRIVER_IO
FoundDPCBoardNumber:
#endif

NWSEndWait(NUTHandle);
NWSEnableInterruptKey(K_AF10, ExitHandler, NUTHandle);

/*
 * Initialize the main options menu.
 */

NWSInitDhList(NUTHandle, Free); /* Don't sort menu items. */

NWSSetDynamicMessage(DYNAMIC_MESSAGE_ONE,
    (BYTE *) "Package Delivery", &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_TWO,
    MSG("Display MLID Stats", 102), &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_THREE,
    MSG("DPC Configuration", 95), &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_FOUR,
    MSG("Dish Pointing", 344), &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_FIVE,
    MSG("Adapter Information", 150), &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_SIX,
    MSG("Modem Control", 501), &NUTHandle->messages);
NWSSetDynamicMessage(DYNAMIC_MESSAGE_SEVEN,

```

```

    MSG("Exit DPCAGENT", 586), &NUTHandle->messages);

if (PackageDelivery)
    NWSAppendToMenu(DYNAMIC_MESSAGE_ONE, 1, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_TWO, 2, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_THREE, 3, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_FOUR, 4, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_FIVE, 5, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_SIX, 6, NUTHandle);
NWSAppendToMenu(DYNAMIC_MESSAGE_SEVEN, 7, NUTHandle);
while (exitFlag == FALSE)
{
    prevChoice = choice;

    /* Set defaultList to previous choice */
    defaultList = NWSGetListHead(NUTHandle);
    if (!PackageDelivery)
        choice -= 1;
    for( i = choice - 1; i; i--)
        defaultList = defaultList->next;

    choice = NWSMenu(InxMSG("DPCAGENT Options", 151),
        10, 40, defaultList, NULL, NUTHandle, NULL);
    if (removedCount != DIORemovedCount)
    {
        NWSDestroyMenu(NUTHandle);
        NWSClearPortal( NUTHandle->portal[mainPortal] );
        goto LookForAdapter;
    }
    switch (choice) {
    case 1:
        if (NWSPushList(NUTHandle)) {
            DisplayPDInterface();
            NWSPopList(NUTHandle);
        }
        break;
    case 2:
        /* Display MLID Stats */
        if (NWSPushList(NUTHandle)) {
            DisplayMLIDStats();
            NWSPopList(NUTHandle);
        }
        break;
    case 3:
        /* Modem Configuration */
        if (NWSPushList(NUTHandle)) {
            DPCConfiguration();
            NWSPopList(NUTHandle);
        }
        break;
    case 4:
        /* Signal Strength Meter */
        if (NWSPushList(NUTHandle)) {
            DPCPointing();
            NWSPopList(NUTHandle);
        }
        break;
    case 5:
        /* Display Adapter Information */
        if (NWSPushList(NUTHandle)) {
            DisplayAdapterInfo();
            NWSPopList(NUTHandle);
        }
    }
}

```

```

    }
    break;

case 6:
    /* Display Adapter Information */
    if (NWSPushList(NUTHandle)) {
        ModemControl();
        NWSPopList(NUTHandle);
    }
    break;

default:
    if (NWSConfirm(InxMSG("Exit DPCAGENT?", 152), 0, 0, TRUE, NULL,
        NUTHandle, NULL) == TRUE)
        exitFlag = TRUE;
    else
        if (choice != 7)
            choice = prevChoice;
    }

    NWSDestroyMenu(NUTHandle);
}

/*****
 *
 * DPCAgentMain(void *parm)
 *
 * Description:
 *   Main thread. It will initialize the NUT screen and wait
 *   for user input.
 *
 * Input:
 *   parm
 *
 * Output:
 *   Nothing
 *
 * Returns:
 *   Nothing
 *
 *****/
void DPCAgentMain(void *parm)
{
    parm = parm;

    MainOptionsHandler();

    ReturnResources(1);
    exit(1);
}

/*****
 *
 * main(int argc, char *argv[])
 *
 * Description:
 *   Initialization routine.
 *
 * Input:
 *   Nothing
 *
 *****/

```

```

 *
 * Output:
 *   Nothing
 *
 * Returns:
 *   0 if successfully initialized
 *
 *****/
LONG ScreenID;
LONG main(int argc, char *argv[])
{
    LONG currentScreen;
    LONG ser[2];
    LONG ccode;
    int i;

    for (i = 1; i < argc; i++)
    {
        if (ICmpB(argv[i], MSG("-DEBUG", 182), 6) == -1)
        {
            if ((DebugFlag = strtoul(argv[i][6], 0, 16)) == 0)
                DebugFlag = TRUE;
        }
    }

    /* Get a handle for allocating a resource tag */
    NLMHHandle = (struct LoadDefinitionStructure *)GetNLMHandle();
    if (!NLMHHandle)
        return(-1);

    if (ReturnMessageInformation((LONG)NLMHHandle, (BYTE **)&NLMMessageTabl
e, NULL, NULL, NULL))
        return(-1);

    OSGetCountryInfo( &GblDOSCountryInfo );

    /* Allocate a resource tag to use for memory allocations */
    allocrTag = AllocateResourceTag(NLMHHandle, TxtMSG("DPCAGENT Memory", 167
),
    ),
    AESTag = AllocateResourceTag(NLMHHandle, TxtMSG("DPCAGENT AES", 168),
        AESProcessSignature);

    asyncrIOTag = AllocateResourceTag(NLMHHandle, TxtMSG("DPCAGENT Async I/O",
169),
    ),
    timerTag = AllocateResourceTag(NLMHHandle, TxtMSG("DPCAGENT Delay Timer",
170),
    ),
    if (allocrTag == NULL ||
        AESTag == NULL ||
        timerTag == NULL ||
        ASYNCRSignature == NULL)
    {
        ConsolePrintf(TxtMSG("DPCAGENT: Unable to allocate Resource Tags
", 171));
        return(-1);
    }

    /* Create a screen for displaying our information */

```

```

currentScreen = GetCurrentScreen();
SetAutoScreenDestructionMode(TRUE);

/* Initialize the screen interface */
ScreenID = CreateScreen("DPCAgent Utility", AUTO_DESTROY_SCREEN);
ccode = NWSInitializeNut(InxMSG("DPC AGENT PROGRAM", 172), AGENT_VERSION

    SMALL_HEADER, NUT_REVISION_LEVEL, NULL, NULL,
    ScreenID, (LONG)allocrTag, &NUTHandle);
    if (ccode)
    {
        ConsolePrintf(TxtMSG("DPCAGENT: Unable to initialize NUT.", 174)
            return(-1);
        }
    }
    // NLMMessageTable = (BYTE **)&(NUTHandle->messages);

/* Get a connection with the server we're on */
if (DebugFlag) {
    DisplayScreen(ScreenID);
    SetCurrentScreen(currentScreen);
}
#ifdef LOG_ECB_ACTIVITY
    if (DebugFlag >= 0x51) {
        DPC_TGID = GetThreadGroupID();
        LogRegisterClient("SYS:DIRECFC/LOG.CFG", 2048, &LogClientHandle);
        LogRegisterEvent("ECB", &LogECBHandle);
    }
#endif /* LOG_ECB_ACTIVITY */
    else {
        DestroyScreen(currentScreen);
    }

#ifdef DEBUG_ALL
    if (OpenScreen(MSG("DPCAGENT Debug Screen", 224), screenTag, &DebugScreenID))
    {
        ConsolePrintf(MSG("DPCAGENT: Unable to create debug screen.", 225));
        return(-1);
    }
#endif

DPCUpdateConfig();

InetChangeProtocol();

DPCSetMaxConnections(ser);

DPCAgentPID = BeginThread(DPCAgentMain, NULL, NULL, NULL);
RenameThread(DPCAgentPID, "DPCAgent Main");
if (PackageDelivery) {
    DPCFilePID = BeginThread(DPCFileMain, NULL, 32 * 1024, NULL);
    RenameThread(DPCFilePID, "DPCAgent PD");
    DPCAccessPID = BeginThread(AccessMain, NULL, NULL, NULL);
    RenameThread(DPCAccessPID, "DPCAgent Access");
}
DPCModemPID = BeginThread(DloMain, NULL, NULL, NULL);
RenameThread(DPCModemPID, "DPCAgent Modem");
if (DPCMaxConnections) {
    DPCInetPID = BeginThread(InetMain, NULL, NULL, NULL);
    RenameThread(DPCInetPID, "DPCAgent Tinet");
}
signal(SIGTERM, ReturnResources);
ExitThread(TSR_THREAD, 0);
}

```

```
ReturnResources(int sig)
```

```
Description:
```

```
Shutdown routine. Returns the modules resources.
```

```
Input: sig
```

```
- ignored
```

```
Output: Nothing
```

```
Returns: Nothing
```

```
void ReturnResources(int sig)
```

```
{
    int i;
    int countdown = MAX_COUNTDOWN;
```

```
sig = sig;
```

```
ExitingFlag = TRUE;
```

```
if (InReturnResources)
    return;
```

```
InReturnResources = TRUE;
```

```
/*
 * Force NUT to escape out of all menus so that it can clean
 * before we call NWSRestoreNUT(NUT has a bug were it will
 * attempt to free up its memory twice(ABEND) if the user
 * leaves the screen a couple of menus in before unloading
 * the application from the command line.
 */
for(i = 0; i < 4; i++)
    NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle,
    InxMSG("Waiting for threads to Exit", 226));

if (AccessAsleep)
    ResumeThread(DPCAccessPID);

if (InetAsleep)
    ResumeThread(DPCInetPID);

/*
 * Give threads and NUT a chance to execute.
 * Wait 1.5 seconds since signal meter and stats could be sleeping
 * for up to a second.
 */
delay(1500);

while(DPCFilePID || DPCModemPID || DPCAccessPID || DPCInetPID) {
    void DPCPDterminate(void);
    DPCPDterminate();

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);
}

```

Thu Jul 17 14:46:11 1997

dpcagent.c

Page 51

```
if (InetAsleep)
    ResumeThread(DPCInetPID);

if (--countdown == 0) {
    Spin(NUTHandle);
    countdown = MAX_COUNTDOWN;
}
ThreadSwitchWithDelay();
}

#ifdef LOG_ECB_ACTIVITY
if (DebugFlag >= 0x51) {
    LogDeregisterEvent(&LogECBHandle);
    LogDeregisterClient(&LogClientHandle);
}
#endif /* LOG_ECB_ACTIVITY */

if (DlCfg.out_protocol == OUT_PPP)
{
    DisconnectPPP();
}
DIODeRegisterAgent();

#ifdef DEBUG_ALL
CloseScreen(DebugScreenID);
#endif
NWSWait(NUTHandle);
NWSRestoreNut(NUTHandle);
}
```

```

#include "dpcagent.h"
/* Our header file */

LONG
int
/*
 * Access Configuration Variables
 */
BYTE SiteID[9];
WORD CDBVersion = 0;
WORD CDBEthVersion = 0;
BYTE DacauFlag = 0;
LONG DacauTime = 0;
LONG RndDacauTime = 0;
DCAURequest_t DacauRequest;
CASDBbuffer_t CASDBpacau;
CASDBbuffer_t CASDBbepb;
CASDBbuffer_t CASDBbdacau;
CASDBbuffer_t CASDBbecau;

/* Elements tables */
static CDBelement_t Elements[MAXELEMENTS];
static CDBelement_t UpdatedElements[MAXELEMENTS];

BYTE *RcvBuf;
MACrecord_t CASmacs[MAX_MAC_RECORDS];
BYTE AccessAddress[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
/* Access thread needs t
o wake up flag */

BYTE
/* Stores current packet
*/
/* ECB linked list variables.
*/
static ECB *AccessECBHead = 0; /* Take ECBs from here */
static ECB *AccessECBTail = 0; /* Put ECBs here */

/* element key used by IroSetAddress.
*/
BYTE MagicKey[] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 };

/* Address Type Table used by MACbuildAddr().
*/
static unsigned char table[5][2] = {
    { MAC_INDIVID, MAC_NORMAL },
    { MAC_INDIVID, MAC_BYPASS },
    { MAC_MULTICAST, MAC_NORMAL },
    { MAC_MULTICAST, MAC_BYPASS }
};

BYTE SerialNum[9];
BYTE SerialNumPacked[3];

/*
 * ReadConfig(void)
 *
 * Description: This routine reads the DPC.CFG file and stores the conte
nts
*/
into the appropriate globale variables.
*/
Input: Nothing
Output: Nothing
Returns: Nothing
*/
*****
static void ReadConfig(void)
{
    int handle, k;
    BYTE *mem_ptr;
    for(k = 0; k < MAXELEMENTS; k++)
    {
        Elements[k].in_use = 'N';
        UpdatedElements[k].in_use = 'N';
    }
    handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 203), O_RDONLY);
    if (handle != -1)
    {
        read(handle, &SiteID, sizeof(SiteID));
        read(handle, &CDBVersion, sizeof(CDBVersion));
        read(handle, &CDBEthVersion, sizeof(CDBEthVersion));
        read(handle, &DacauFlag, sizeof(DacauFlag));
        read(handle, &DacauTime, sizeof(DacauTime));
        read(handle, &DacauRequest, sizeof(DCAUrequest_t));
        read(handle, &CASDBpacau, sizeof(CASDBbuffer));
        read(handle, &CASDBbepb, sizeof(CASDBbuffer));
        read(handle, &CASDBbdacau, sizeof(CASDBbuffer));
        read(handle, &CASDBbecau, sizeof(CASDBbuffer));
    }
    else
        UpdateFileStatus(MSG("Obtaining Encryption Keys", 204));
    for(k = 0; k < MAX_MAC_RECORDS; k++)
        CASmacs[k].in_use = 'N';
    if (handle != -1)
    {
        close(handle);
    }
    else
    {
        SiteID[0] = '\0';
        CASDBpacau.version = CASDBbepb.version = 0;
        CASDBbdacau.version = CASDBbecau.version = 0;
        CASDBpacau.entries = CASDBbepb.entries = 0;
        CASDBbdacau.entries = CASDBbecau.entries = 0;
    }
    CASDBpacau.entry_len = PACAU_LEN;
    CASDBbepb.entry_len = PEB_LEN;
    CASDBbdacau.entry_len = DACAU_LEN;
    CASDBbecau.entry_len = ECAU_LEN;
    if (DacauFlag)
    {
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 257));
        WaitingForKeys = TRUE;
    }
}

```



```

DacauRequest.opcode = DACAU_REQUEST;
RndDacauTime = time(0) + rand();
)

```

```

/*****

```

```

* SaveConfig(void *parm)

```

```

*
* Description: This routine writes the access structures out to DPC.CFG
* anytime a change is made to any of the structures.

```

```

*
* Input: Nothing
*
* Output: Nothing
*
* Returns: Nothing

```

```

/*****

```

```

static void SaveConfig(void)

```

```

{
    int handle;
    int tmpFlag;

```

```

    handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 206), O_RDWR | O_CREAT, S_
    IWRITE | S_IREAD);

```

```

    tmpFlag = DacauFlag;
    if (WaitingForKeys)
    {

```

```

        DacauFlag = 1;
    }

```

```

    /* Write version */

```

```

    write(handle, SiteID, sizeof(SiteID));
    write(handle, &CDRVersion, sizeof(CDRVersion));
    write(handle, &CDBETHVersion, sizeof(CDBETHVersion));
    write(handle, &DacauFlag, sizeof(DacauFlag));
    write(handle, &DacauTime, sizeof(DacauTime));
    write(handle, &DacauRequest, sizeof(DACAUrequest_t));

```

```

    /* Write Info headers */

```

```

    write(handle, &CASDDBpacau, sizeof(CASDDBbuffer));
    write(handle, &CASDDBpeb, sizeof(CASDDBbuffer));
    write(handle, &CASDDBdacau, sizeof(CASDDBbuffer));
    write(handle, &CASDDBecau, sizeof(CASDDBbuffer));

```

```

    /* Write buffers */
    close(handle);

```

```

    DacauFlag = tmpFlag;

```

```

/*****

```

```

* AccessESR(ECB *ecb)

```

```

*
* Description:

```

```

    This routine is called by the MLID interrupt service
    routine when a packet is received. The ECB is queued
    and if the Access File thread is sleeping, it is

```

```

    woken up.

```

```

* Input: ecb

```

```

* the packet - pointer to the ECB describing

```

```

* Output: nothing

```

```

* Returns: 0

```

```

*
* We always keep the ECB
/*****

```

```

int AccessESR(ECB *ecb)

```

```

{

```

```

    /* Link the ECB to the Tail of the linked list */
    ecb->ECB_NextLink = 0;
    if (AccessESRTail)

```

```

        AccessESRTail->ECB_NextLink = ecb;

```

```

    AccessESRTail = ecb;

```

```

    if (AccessESRHead == 0)

```

```

        AccessESRHead = ecb;

```

```

    /* Wake up file thread only if we need to */
    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

```

```

#ifdef LOG_ECB_ACTIVITY

```

```

    if (LogECBHandle) {

```

```

        int TGID = SetThreadGroupID(DPC_TGID);

```

```

        LogMsg(LogClientHandle, LogECBHandle, FALSE,

```

```

            "ACCESS queue(%08lx)\n", ecb);

```

```

        SetThreadGroupID(TGID);
    }

```

```

#endif /* LOG_ECB_ACTIVITY */

```

```

    return(0);
}

```

```

/*****

```

```

* find_peb(ID element_pattern,
* char ver_pattern)

```

```

*
* Description:

```

```

    This routine searches the PEB list to find an entry that
    matches the element and version pattern passed in.

```

```

* Input:

```

```

    element_pattern
    ver_pattern

```

```

to search for

```

```

* Output: nothing

```

```

* Returns:

```

```

    pointer to peb entry if it was found
    otherwise its a NULL

```

```

/*****

```

```

MUXecau_t *find_ecau(ID group_pattern, char ver_pattern)

```

```

{

```

```

int i;
MUXecau_t *p_ecau, *ret = NULL;

for(i = 0; i < CASDBecau.length; i += ECAU_LEN)
{
    p_ecau = (MUXecau_t *) (CASDBecau.p_buffer + i);
    if(CmpB(&p_ecau->groupid, &group_pattern, sizeof(ID)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_ecau;
            break;
        }
        else
        {
            if(ver_pattern == p_ecau->version)
            {
                ret = p_ecau;
                break;
            }
        }
    }
    return(ret);
}

```

```

static MUXpeb_t *find_peb(ID element_pattern, char ver_pattern)
{

```

```

    unsigned short i, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;

    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        if(CmpB(&p_peb->elementid, &element_pattern, sizeof(ID)) == -1)
        {
            if(ver_pattern == -1)
            {
                ret = p_peb;
                break;
            }
            else
            {
                if(ver_pattern == p_peb->version)
                {
                    ret = p_peb;
                    break;
                }
            }
        }
        num_addr = p_peb->num_addr[0];
        num_addr |= ((unsigned short) p_peb->num_addr[1]) << 8;
        i += num_addr * MAC_LENGTH;
    }
    return(ret);
}

```

```

/*
 * Deletes element not only from the CASDB
 * but from adapter as well
 */

```

```

/*****
 *
 * del_peb_element(int e_num)
 */

```

```

*
* Description:
*
* This routine deletes and from the CASDB and from the
* adapter.
*
* Input:
*
* e_num
*
* delete
*
* Output:
*
* nothing
*
* Returns:
*
* 0
*
* *****
*
* static del_peb_element(int e_num)
*
* {
*
*     int k;
*
*     DIDeleteAddress(Elements[e_num].channel, (BYTE *) &Elements[e_num].e_mac
*
*     for(k = 0; k < MAX_MAC_RECORDS; k++)
*     {
*         if(CASmacs[k].in_use == 'Y' &&
*            CmpB(&CASmacs[k].dpc_mac, &Elements[e_num].e_mac, sizeof(MACadd
*
*         r_t)) == -1)
*         {
*             CASmacs[k].in_use = 'N';
*             break;
*         }
*     }
*
*     Elements[e_num].in_use = 'N';
*     return(0);
*
* /*****
*
*     replace_peb_element(int num,
*                          unsigned char new_ver)
*
* Description:
*
* This routine replaces the version numbers of the element
* indexed by num.
*
* Input:
*
* int num
*
* int to modify
*
* new_ver
*
* stuff into Element entry
*
* Output:
*
* nothing
*
* Returns:
*
* 0
*
* *****
*
* static replace_peb_element(int num, unsigned char new_ver)
*
* {
*
*     int k;

```

```

- Index of Eleme
- new version to

```

```

for(k = 0; k < MAX_MAC_RECORDS; k++)
{
    if(CASmacs[k].in_use == 'Y' &&
       CCmpB(&CASmacs[k].dpc_mac, &Elements[num].e_mac, sizeof(MACAddr_
t)) == -1)
    {
        CASmacs[k].dpc_mac.Ver = new_ver;
        break;
    }
    Elements[num].e_mac.Ver = new_ver;
    Elements[num].e_ver = new_ver;
    return 0;
}

/*****
*
* find_pacau(ID group_pattern,
*           char ver_pattern)
*
* Description: This routine attempts to locate a pacau given a group
*              pattern.
*
* Input:      group_pattern
*              ver_pattern
*              - group pattern to match
*              - version portion of the
*              pattern
*
* Output:
*
* Returns:    pointer to pacau if successful
*              Otherwise NULL
*
* *****/
MUXpacau_t *find_pacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXpacau_t *p_pacau, *ret = NULL;
    for(i = 0; i < CASDBpacau.length; i += PACAU_LEN)
    {
        p_pacau = (MUXpacau_t *) (CASDBpacau.p_buffer + i);
        if((CCmpB(&p_pacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                break;
            }
            else
            {
                if(ver_pattern == p_pacau->version)
                {
                    ret = p_pacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
*
* reverse_key(BYTE *key)
*
* Description: This routine swaps the byte order of the 8 byte
*              key passed in.
*
* Input:      key
*
* Output:
*
* Returns:    pointer to pacau if successful
*              Otherwise NULL
*
* *****/

```

```

*
* find_dacau(ID group_pattern,
*           char ver_pattern)
*
* Description: This routine attempts to locate a dacau given a group
*              pattern.
*
* Input:      group_pattern
*              ver_pattern
*              - group pattern to match
*              - version portion of the
*              pattern
*
* Output:
*
* Returns:    pointer to dacau if successful
*              Otherwise NULL
*
* *****/
MUXdacau_t *find_dacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXdacau_t *p_dacau, *ret = NULL;
    for(i = 0; i < CASDBdacau.length; i += DACAU_LEN)
    {
        p_dacau = (MUXdacau_t *) (CASDBdacau.p_buffer + i);
        if((CCmpB(&p_dacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                break;
            }
            else
            {
                if(ver_pattern == p_dacau->version)
                {
                    ret = p_dacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
*
* reverse_key(BYTE *key)
*
* Description: This routine swaps the byte order of the 8 byte
*              key passed in.
*
* Input:      key
*
* Output:
*
* Returns:    pointer to pacau if successful
*              Otherwise NULL
*
* *****/

```

```

Thu Jul 17 14:46:11 1997
*****
access.c
*****
Page 9 Thu Jul 17 14:46:11 1997
*****
access.c
*****
Page 10
*****
- pointe

void reverse_key(BYTE *key)
{
    unsigned char x;

    x = key[0]; key[0] = key[1]; key[1] = x;
    x = key[2]; key[2] = key[3]; key[3] = x;
    x = key[4]; key[4] = key[5]; key[5] = x;
    x = key[6]; key[6] = key[7]; key[7] = x;
}

/*****
*
* make_element_id(BYTE *e_id,
* char *e_id_txt)
*
* Description: This routine is called by LroSetAddress to help
* build the element id.
*
* Input: e_id
* element id
*
* Output: e_id_txt
* element text
*
* Returns: nothing
*
*****/

void make_element_id(BYTE *e_id, char *e_id_txt)
{
    BYTE work[3];
    static char HexChar[] = MSG("0123456789ABCDEF", 208);
    unsigned char Ch, *p;
    int count = 3, i = 0;

    work[0] = e_id[2];
    work[1] = e_id[1];
    work[2] = e_id[0];

    p = work;
    while (count--)
    {
        Ch = *p++;
        e_id_txt[i++] = HexChar[Ch>>4]; /* high nibble */
        e_id_txt[i++] = HexChar[Ch & 0x0f]; /* low nibble */
    }
    e_id_txt[i] = '\0';
}

/*****
*
* int43(LONG id, BYTE *array)
*
* Description: This routine is called by MACbuildAddr to convert
* an id to its 3 byte equivalent.
*
* Input: id
* id to convert
*
*****/

int MACbuildAddr(char *element_txt, int feature, BYTE ver, MACAddr_t *address)
{
    LONG i;
}

/*****
*
* MACbuildAddr(char *element_txt,
* int feature,
* BYTE ver,
* MACAddr_t *address)
*
* Description: This routine is called by LroSetAddress to build a
* MAC address out of a file_id and community id.
*
* Input: element_txt
* feature
* AC_PKG, MAC_DF
* _BYPASS_MULTICAST
* ver
* community id
* address
*
* Output: address is filled in
*
* Returns: 0 if successful
*
*****/

int MACbuildAddr(char *element_txt, int feature, BYTE ver, MACAddr_t *address)
{
    LONG i;
}

```



```

return ret;
)
/*****
*
* hextoi(int c)
*
* Description:
*   Converts an ASCII hex character into an integer.
*
* Input:
*   c
*
* character
*
* Output:
*   Nothing
*
* Returns:
*   -1 if not hex character
*   otherwise its the integer equivalent
*
*****/
static int hextoi(
    int c)
{
    char digit, lower, upper;
    digit = (c >= '0' && c <= '9');
    lower = (c >= 'a' && c <= 'f');
    upper = (c >= 'A' && c <= 'F');

    if (digit)
        return (c - '0');
    if (lower)
        return (c - 'a' + 10);
    if (upper)
        return (c - 'A' + 10);
    return (-1);
}
/*****
*
* pack_mac_addr(BYTE *packed_address,
*               int packed_address_len,
*               BYTE *address,
*               int address_len)
*
* Description:
*   Converts a character string containing the mac address i
*   nto
*   packed BCD digits.
*
* Input:
*   packed_address_len - length of the packed buffer
*   address            - Hex ASCII string to co
*   address_len       - length of the hex ASCII
*
* I string
*
* Output:
*   packed_buffer - buffer to write packed address
*
* into.
*
*****/
Returns:
    TRUE if packed successfully
*
*****/
#define LEFT 0
#define RIGHT 1
int pack_mac_addr( BYTE *packed_address, int packed_address_len,
    BYTE *address, int address_len)
{
    BYTE c, side;
    int i, j;
    /*
    * Pack hex digit ascii string in "address" into binary in
    * "packed_address". Return FALSE if unsuccessful. If number of hex
    * digits in "address" cannot fill "packed_address", then
    * "packed_address" is padded to the right with zeros.
    */
    side = LEFT;
    for (i = 0; i < packed_address_len; i++)
        packed_address[i] = 0;
    for (i = 0, j = 0; i < address_len; i++)
    {
        if ((c = hextoi(address[i])) == 0xff)
            return (FALSE);
        if (side == LEFT)
        {
            c = c << 4;
        }
        packed_address[j] |= c;
        if (++side > RIGHT)
        {
            side = LEFT;
            if (++j > packed_address_len)
                return (FALSE);
        }
    }
    return (TRUE);
}
/*****
*
* check_df_groups(void)
*
* Description:
*   This function is called when new PACAU or DACAU is proce
*   ssing.
*   It checks if there are any changes in groups membership
*   to avoid receiving DF elements when membership of this a
*   to the proper DF group has been deleted.
*
* Input:
*   Nothing
*
* Output:
*   Nothing
*
* Returns:
*   0 if check was successful
*
*****/

```

```
static check_df_groups(void)
```

```
{
    int i;
    MUXpacau_t *pacau;
    MUXdcau_t *dcau;
    MUXpeb_t *p_peb;
```

```
    for(i = 0; i < MAXELEMENTS; i++) {
        if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
            continue;
        if((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            continue;
        pacau = find_pacau(p_peb->groupid, p_peb->version);
        dcau = find_dcau(p_peb->groupid, p_peb->version);

        if(dcau == NULL && pacau == NULL) {
            /* We have no group for this element */
            del_peb_element(i);
        }
    }
    return 0;
}
```

```
/* ***** */
```

```
    parse_pacau(BYTE *buf, WORD len)
```

```
    Description:
```

```
    Parse the PACAU packet. This is where we detect that we
    must receive a new key via the modem(packets d version
    will not match CASDBdcau.version.
```

```
    Input:    buf
```

```
              - pointer to the message received
```

```
    len
```

```
              - length of the message received
```

```
    Output:    Nothing
```

```
    Returns:    0 if successfully parsed
```

```
/* ***** */
```

```
static parse_pacau(BYTE *buf, WORD len)
```

```
{
    LONG curr_p_version;
    LONG curr_d_version;
    LONG curr_d_time;
    int ret = 0, k;
```

```
    if(strncmp(buf, SiteID, 8) != ESUCCESS) {
        strncpy(SiteID, buf, 8);
        SiteID[8] = 0;
        CDBVersion++;
        /* New Site ID - reset versions of PACAU, DACAU, ...*/
        CASDBpacau.version = CASDBpeb.version = 0;
        CASDBdcau.version = CASDBbecau.version = 0;
        SaveConfig();
    }
```

```
    curr_p_version =
```

```
    buf[8] +
    buf[9] * 256UL +
    buf[10] * 256UL * 256UL +
    buf[11] * 256UL * 256UL * 256UL;
```

```
    curr_d_version =
```

```
    buf[12] +
    buf[13] * 256UL +
    buf[14] * 256UL * 256UL +
    buf[15] * 256UL * 256UL * 256UL;
```

```
    curr_d_time =
```

```
    buf[16] +
    buf[17] * 256UL +
    buf[18] * 256UL * 256UL +
    buf[19] * 256UL * 256UL * 256UL;
```

```
    if(curr_d_version != CASDBdcau.version) {
        /* There are some changes in the DACAU stuff */
        /* Build DACAU request */
        DcauFlag = 1;
        srand(time(0));
        RndDcauTime = curr_d_time;
        RndDcauTime = time(0) + rand();
        CDBVersion++;
    }
```

```
    WaitingForKeys = TRUE;
```

```
    UpdateFileStatus(MSG("Obtaining Encryption Keys", 138));
```

```
    memcpy(&DcauRequest.d_version, buf + 8 + sizeof(VER), sizeof(VER));
#ifdef USE_NEW_MIPS_CODE
```

```
    DcauRequest.d_version.i[3] |= 0x80;
```

```
#endif
```

```
    CASDBdcau.version = curr_d_version;
```

```
    SaveConfig();
}
```

```
    if(curr_p_version != CASDBpacau.version) {
        CDBVersion++;
```

```
        CASDBpacau.version = curr_p_version;
        CASDBpacau.length = len - PACAU_HEAD_LEN;
        memcpy(CASDBpacau.p_buffer,
            buf + PACAU_HEAD_LEN,
            CASDBpacau.length);
```

```
        CASDBpacau.entries = CASDBpacau.length / CASDBpacau.entry_len;
        check_df_groups();
        SaveConfig();
    }
```

```
    return ret;
}
```

```
/* ***** */
```

```
    parse_ecau(BYTE *buf, WORD len)
```

```
    Description:
```

```
    Parse the ECAU packet. Replace the old entry by the
    new one as long as version doesn't match CASDBbecau versi
```

```
on.
```

```
    Input:    buf
```

```
              - pointer to the message received
```

```
    len
```

```
              - length of the message received
```

```

*
* Output: Nothing
*
* Returns: 0 if successfully parsed
*
*****
static parse_ecau(BYTE *buf, WORD len)
{
    LONG curr_e_version;
    int ret = 0;

    curr_e_version =
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +
        buf[3] * 256UL * 256UL * 256UL;

    if (curr_e_version != CASDBecau.version)
    {
        CDBVersion++;
        CASDBecau.version = curr_e_version;
        CMovB(buf + ECAU_HEAD_LEN, CASDBecau.p_buffer, len - ECAU_HEAD_LEN);
        CASDBecau.length = len - ECAU_HEAD_LEN;
        CASDBecau.entries = CASDBecau.length / CASDBecau.entry_len;
        SaveConfig();
    }
    return ret;
}

/*****
*
* parse_peb(BYTE *buf, WORD len)
*
* Description: Parse the PEB packet. Replace the old entry by the
*              new one. Check elements and add new addresses to the
*              MLID and delete old ones.
*
* Input: buf - pointer to the message received
*        len - length of the message received
*
* Output: Nothing
*
* Returns: 0 if successfully parsed
*
*****
static parse_peb(BYTE *buf, WORD len)
{
    int i, k, macs, not_found;
    MUXpeb_t *p_peb;
    unsigned long curr_version;
    int ret = 0;
    MUXpacau_t *pacau;
    MUXdacau_t *dacau;

    curr_version =
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +

```

```

        buf[3] * 256UL * 256UL * 256UL;

    if (curr_version != CASDBpeb.version)
    {
        CDBVersion++;
        CASDBpeb.version = curr_version;
        CMovB(buf + PEB_HEAD_LEN, CASDBpeb.p_buffer, len - PEB_HEAD_LEN);
        CASDBpeb.length = len - PEB_HEAD_LEN;
        CASDBpeb.entries = CASDBpeb.length / CASDBpeb.entry_len;
        /* Walk through the elements table and check ... */
        for (i = 0; i < MAXELEMENTS; i++)
        {
            if (Elements[i].in_use == 'N' || Elements[i].packfeed != 'P')
                continue;
            if ((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            {
                /* Element no longer valid: Delete. */
                del_peb_element(i);
                continue;
            }
            if (p_peb->version != Elements[i].e_ver)
            {
                /* We have found element but with different version: */
                /* It means, that we somehow miss key update */
                /* Replace inside adapter and in CDB. */
                /* Delete address */
                DIODeleteAddress(Elements[i].channel, (BYTE *) &E
                lements[i].e_mac);

                /* Replace element in the CDB */
                replace_peb_element(i, p_peb->version);
                /* Trying to Add address */
                /* At first find group for the element */
                pacau = find_pacau(p_peb->groupid, p_peb->version);
                dacau = find_dacau(p_peb->groupid, p_peb->versio
                n);

                if (dacau != NULL)
                {
                    pacau = (MUXpacau_t *) dacau;
                    if (pacau != NULL)
                    {
                        if (DIOAddGroupAddress(Elements[i].chann
                        el, (BYTE *) &Elements[i].e_mac,
                        (BYTE *) &pacau->g_key))

                            /* Find a group, Add */
                            channelCfg.CfgChannel = Elements[i].chan
                            nel;
                            channelCfg.CfgNumAddresses = 1;
                            CMovB(&Elements[i].e_mac, channelCfg.Cfg
                            CMovB(&pacau->g_key, key, 8);
                            reverse_key(&key);
                            CMovB(key, channelCfg.CfgGroupKey, 8);
                            CMovB(&MagicKey, channelCfg.CfgElementKe
                            y, 8);

                            ecb.ECB_StackID = MLID_ADD_ADDRESS;
                            ecb.ECB_Fragment[0].FragmentAddress = &c
                            if (IoctlMlId(FDBBoard, &ecb, FDBControl
                            del_peb_element(i);
                    }
                }
            }
        }
    }
}

```



```

    else
    {
        /* Can't find group for this element */
        /* We are not subscribed on this group any more */
        /* Delete element. */
        del_peb_element(i);
    }

    /* Check Ethernet addresses within Element. */
    macs = not_found = 0;
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CCompB(&CASmacs[k].dpc_mac, &Elements[i].e_mac, s
           izeof(MACaddr_t)) == -1)
        {
            macs++;
            if(find_peb_mac(CASmacs[k].e_mac) == NULL)
            {
                /* Somebody in the NOC has deleted Ethernet */
                /* address of this element... */
                not_found++;
                CASmacs[k].in_use = 'N';
            }
        }
        if(macs == not_found)
        {
            /* There are no ethernet addresses for the element */
            del_peb_element(i);
        }
    }
    SaveConfig();
    return ret;
}

/*****
 *
 * parse_gup(BYTE *buf, WORD len)
 *
 * Description: Parse the GUP packet. Add new entries to the PACAU buffe
 *
 * Input: buf
 *         - pointer to the message receive
 *         len
 *         - length of the message received
 *
 * Output: Nothing
 *
 * Returns: 0 if successfully parsed
 *
 *****/
static parse_gup(BYTE *buf, WORD len)
{
    GUPid_t *p_gup_element;
    GUPhead_t *p_gup_head;
    MUXpacau_t *p_pacau;
    int i, curr_len = 0;

    if
    {
        static update_peb(BYTE *buf, WORD len)
        {
            int m;
            MUXpacau_t *pacau = NULL;
            MUXdacau_t *dacau = NULL;

            Returns: 0 if successfully parsed

            Output: Nothing
                len
                - length of the message received
            Input: buf
                - pointer to the message receive
            Description: Parse the UPDATE_PEB packet. Replace the old entry by th
                new one according to the Element ID.
            update_peb(BYTE *buf, WORD len)
        }
    }
    return 0;
}

ngth);
    P_pacau_tmp = (MUXpacau_t *) (CASDBpacau.p_buffer + CASDBpacau.le
    ngth);
    CASDBpacau.length += PACAU_LEN;
    CASDBpacau.entries++;
    CMovB(&p_gup_head->groupid, &p_pacau_tmp->groupid, sizeof(ID));
    p_pacau_tmp->version = p_gup_head->g_ver;
    CMovB(&p_gup_element->g_key, &p_pacau_tmp->g_key, sizeof(chunk));
    break;
}
}
}
return 0;
}

/*****
 *
 * update_peb(BYTE *buf, WORD len)
 *
 * Description: Parse the UPDATE_PEB packet. Replace the old entry by th
 *
 * Input: buf
 *         - pointer to the message receive
 *         len
 *         - length of the message received
 *
 * Output: Nothing
 *
 * Returns: 0 if successfully parsed
 *
 *****/
static update_peb(BYTE *buf, WORD len)
{
    int m;
    MUXpacau_t *pacau = NULL;
    MUXdacau_t *dacau = NULL;
}

```

```

MUXpeb_t *old_peb, *new_peb, *found;
int found_element;
short ret_code = 0;
unsigned long curr_version;

if (len != 2 * PEB_LEN + PEB_HEAD_LEN)
    return(0);
curr_version =
    buf[0] +
    buf[1] * 256UL +
    buf[2] * 256UL * 256UL +
    buf[3] * 256UL * 256UL * 256UL;

if (curr_version == CASDBpeb.version)
    return 0;
CDBVersion++;
CASDBpeb.version = curr_version;

old_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN);
new_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN + PEB_LEN);
if ((found = find_peb(old_peb->elementid, old_peb->version)) == NULL)
    /* Nothing to replace - ERROR */
    return(0);
for (m = 0; m < MAXELEMENTS; m++)
{
    if (UpdatedElements[m].in_use == 'Y')
    {
        /* We have received next replace element command,
         * it means, that the previous element
         * has already been updated at the
         * DataFeed Lan Gateway and
         * we can delete old address and keys
         * from adapter
         */

        ret_code = DIDeleteAddress(UpdatedElements[m].channel,
            (BYTE *)&UpdatedElements[m].e_mac);

        UpdatedElements[m].in_use = 'N';
    }
}
/* Has been the element loaded before? */
found_element = find_element_id(old_peb->elementid, old_peb->version);
if (found_element != -1)
{
    /* Yes. We are receiving this element now.
     * Let's keep old element's address
     * in the UpdatedElement table
     */
    CMovB(&Elements[found_element], &UpdatedElements[found_element],
        sizeof(CDBelement_t));

    /* Trying to find a group for the element
     */
    pacau = find_pacau(new_peb->groupid, new_peb->grversion);
    dacau = find_dacau(new_peb->groupid, new_peb->grversion);
    if (dacau != NULL)
        pacau = (MUXpacau_t *) dacau;
        if (pacau != NULL)
        {
            /* Put element in the adapter
             * After this operation in the adapter will be
             * both old and new element's addresses and keys
             */
        }
    }
    else
    {
        /* No group, Delete element */
        del_peb_element(found_element);
    }
    /* Change PEB database for this element */
    if (!ret_code)
    {
        replace_peb_element(found_element, new_peb->version);
        CMovB( (unsigned char *)new_peb, (unsigned char *)found, PEB_LEN - 2);
        return(ret_code);
    }
}
/*****
 * AccessReceive(char *message)
 *
 * Description: This routine checks to see if we've received any
 * packets from the MLID. If we have, the data is copied
 * from the ECB to the message and the ECB is returned to t
 * he
 * LSL.
 *
 * Input: message
 * to where to copy data
 *
 * Output: message and lroinfo filled in if successful
 *
 * Returns: 0 if a packet has been received
 *
 *****/
LONG AccessReceive(char *message)
{
    ECB *ecb;

    /* Extract an ECB from the linked list if one exists */
    Disable();
    ecb = AccessECBHead;
    if (!ecb)
    {
        /* No ecb. Just return */
        Enable();
        return(-1);
    }
    AccessECBHead = ecb->ECB_NextLink;
    if (AccessECBHead == 0)
        AccessECBTail = 0;
    Enable();

    /* copy the data past the lroinfo to the message */
    CMovB(ecb->ECB_Fragment[0].FragmentAddress, message, ecb->ECB_Fragment[0].
        FragmentLength);

    /* return the ecb to the LSL */
}

```

```

    CUSLReturnRCvECB(ecb);
#ifdef LOG_ECB_ACTIVITY
    FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
        "ACCESS return(%08lx)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
    return(0);
}

/*****
 *
 * AccessAdd(BYTE *message)
 *
 * Description:
 *     This routine is called when a packet is received
 *     and is responsible for dispatching it.
 *
 * Input:
 *     message
 *     lroinfo
 *     - packet data
 *     - lro information
 *
 * Output:
 *     Nothing
 *
 * Returns:
 *     Nothing
 *
 *****/
int AccessAdd(BYTE *message) {
    IndPacket_t *p_packet;
    int packet_type;
    int status;

    p_packet = (IndPacket_t *)message;

    if(p_packet->address[3] == 0x01)
        switch(p_packet->payload(0)) {
            case SG_PACAU:
                packet_type = PACAU;
                break;
            case SG_ECAU:
                packet_type = ECAU;
                break;
        }
    else if(p_packet->address[0] == 0x03) { /* Bypass key */
        switch(p_packet->payload(0)) {
            case PEB_PACKET:
                packet_type = PEB;
                break;
            case GUP_PACKET:
                packet_type = GUP;
                break;
            case PEB_UPDATE_PACKET:
                packet_type = PEB_UPDATE;
                break;
            default:
                packet_type = UNKNOWN;
                break;
        }
    }
    else
        packet_type = UNKNOWN;

    switch(packet_type) {
        case UNKNOWN:
            break;
    }
}

```

```

case PACAU:
    status = parse_pacau(p_packet->payload+1, p_packet->length-1);
    break;
case ECAU:
    status = parse_eacau(p_packet->payload+1, p_packet->length-1);
    break;
case PEB:
    status = parse_peb(p_packet->payload+1, p_packet->length-1);
    break;
case GUP:
    status = parse_gup(p_packet->payload+1, p_packet->length-1);
    break;
case PEB_UPDATE:
    status = update_peb(p_packet->payload+1, p_packet->length-1);
    break;
}
return(status);
}

/*****
 *
 * parse_dacau(BYTE *buf,
 *             int len)
 *
 * Description:
 *     Parse the new DACAU from the message received by the mod
 *     Replace the old DACAU buffer by the new one if the the v
 *     matches what we have in the CASDBdacau version.
 *
 * Input:
 *     buf
 *     len
 *
 * Output:
 *     Nothing
 *
 * Returns:
 *     0 if successful
 *
 *****/
static int parse_dacau(BYTE *buf, int len)
{
    unsigned long curr_d_version;
    int ret = 0;

    curr_d_version =
        buf[0] * 256UL +
        buf[1] * 256UL +
        buf[2] * 256UL +
        buf[3] * 256UL * 256UL;

    if(curr_d_version == CASDBdacau.version) {
        CASDBversion++;
        CASDBdacau.version = curr_d_version;
        CMovB(buf + DACAU_HEAD_LEN, CASDBdacau.p_buffer, len - DACAU_HEAD_LEN);
        CASDBdacau.length = len - DACAU_HEAD_LEN;
        CASDBdacau.entries = CASDBdacau.length / CASDBdacau.entry_len;
        check_df_groups();
    }
    else {
        ret = -1;
    }
}

```

```

}
return ret;
}

/*****
 *
 * DacauResponse(BYTE *pdata,
 *               int len,
 *               timeoutFlag)
 *
 * Description:
 *
 * This is the callback routine pass in to DloScheduleRecei
 * which receives the response from the modem. If a timeout
 * hadn't occurred, parse the message for the dacau info.
 *
 * Input:
 *   pdata      - Pointer to response me
 *   len        - length of the message
 *   timeoutFlag - non-zero if we timed out
 *
 * Output:
 *   Nothing
 *
 * Returns:
 *   Nothing
 *
 * *****/
static void DacauResponse(BYTE *pdata, int len, int timeoutFlag)
{
    if (timeoutFlag)
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 258));
        return;
    }
    EnterDebugger();
    if (parse_dacau(pdata, len) == 0)
    {
        WaitingForKeys = FALSE;
        CDBVersion++;
        SaveConfig();
        UpdateFileStatus(MSG("IDLE", 211));
    }
    else
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 259));
    }
}

/*****
 *
 * GetDacau(void)
 *
 * Description:
 *
 * Initiate the reception of the encryption keys for this a
 * by constructing a request, sending it to the modem, and
 * scheduling a receive routine to receive the response whi
 * hopefully contains our encryption key.
 *
 * Input:
 *
 * *****/
void AccessMain(void *parm)
{
    LONG sn, ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 212)};
    ChannelConfig channelCfg;
    BYTE address[8];
    BYTE x;
    LONG removedCount;
}

```

```

parm = parm;

/*
 * When MLID has been found, Open the conditional access channel.
 */

RegisterWithDriver:
while(!ExitingFlag)
(
    AccessAsleep = TRUE;
    delay(500);
    AccessAsleep = FALSE;

    if (DIOBoard != 0)
    (
        removedCount = DIORemovedCount;
        if (AccessChannel != -1 ||
            DIOOpenChannel(AccessAddress, AccessESR,
                &AccessChannel) == 0)
        (
            DIOGetSN(SerialNum);
            sn = atol(SerialNum);
            NMSprintf(SerialNum, MSG("%06lX", 213), sn);

            pack_mac_addr(SerialNumPacked, 3, SerialNum, 6);
            x = SerialNumPacked[0];
            SerialNumPacked[0] = SerialNumPacked[2];
            SerialNumPacked[2] = x;

            MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACaddr
                _t *)address);

            if (DIOAddAddress(AccessChannel, address) == 0)
            {
                channelCfg.CfgChannel = AccessChannel;
                channelCfg.CfgNumAddresses = 1;
                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACaddr
                    _t *)channelCfg.CfgAddress);

                ecb.ECB_StackID = MLID_ADD_ADDRESS;
                ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
                if (IoctlMlid(FDBBoard, &ecb, FDBControlEntry) ==
                    0)
                {
                    break;
                }
            }

            if (ExitingFlag)
            (
                DPCAccessPID = 0;
                return;
            )

            /*
             * Now lets open up the the DPC.CFG file.
             */

            ReadConfig();
            while(!ExitingFlag)
            (

```

```

        ccode = AccessReceive(AccessMessage);
        if (ccode)
        (

```

```

            AccessAsleep = TRUE;
            SuspendThread(DPCAccessPID);
            AccessAsleep = FALSE;
            if (removedCount != DIORemovedCount)
                goto RegisterWithDriver;
            continue;
        )
    )

```

```

    AccessAdd(AccessMessage);
    if (DacauFlag)
        GetDacau();
}

```

```

DIOCloseChannel(AccessChannel);
DPCAccessPID = 0;

```

```

#include "dpcagent.h" /* Our header file */

LONG DIOBoard = 0;
LONG DIORemovedCount = 0;
LONG DIOControlEntry = 0;
struct DriverStatsStructure* DIOStats = 0;

void DIORemove(void)
{
    int i;

    /* Invalidate the DriverIO board. Increment removed count so that
     * other threads can detect that the driver has changed, even if
     * DIOBoard gets filled in. The other threads can then re-register
     * with the new driver.
     */

    DIOStats = 0;
    DIOBoard = 0;
    DIORemovedCount++;

    /* Force the NWSNUT menus to exit so that DPCAGENT can go back
     * to searching for a new adapter before allowing user to choose
     * options.
     */

    for(i = 0; i < 4; i++)
        NWSUngetkey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

    /* Force sleeping threads to wake up so that they can detect that
     * the adapter has been unloaded(DIORemovedCount will be different
     * from their local copy of the last removed count). The threads
     * should then spin(sleep) periodically until a new adapter is
     * present, and then re-register with the adapter if they need to.
     */

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

    if (InetAsleep)
        ResumeThread(DPCInetPID);

    LONG DIORRegisterWithAdapter(char *shortName)
    {
        LONG board;

        for(board = 0; board < NumberOfLANs; board++)
        {
            void (*ControlEntryPoint) (void) = NULL;
            if (CUGLGetMLIDControlEntry(board,
                                         &ControlEntryPoint) == ESUCCESS)
            {
                struct DriverConfigurationStructure* config = 0;
                if ((config = (struct DriverConfigurationStructure *) Co
                    mmAndMlid(board, 0, (LONG)ControlEntryPoint)))
                {
                    if (!CStrCmp(config->DShortName, shortName))
                    {
                        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC",
505));

```

```

void (*removeRoutine) () = DIORemove;

ecb.ECB_StackID = MLID_REGISTER_AGENT;
ecb.ECB_Fragment[0].FragmentAddress = &r

/* Call MLID */
IoctlMlid(board, &ecb, (LONG)ControlEntr

DIOBoard = board;
DIOControlEntry = (LONG)ControlEntryPoin
return(0);

        )
        return(-1);
    )

void DIORRegisterAgent(void)
{
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 506)};
    void (*nullRoutine) () = 0;
    if (DIOBoard)
    {
        ecb.ECB_StackID = MLID_REGISTER_AGENT;
        ecb.ECB_Fragment[0].FragmentAddress = &nullRoutine;

        /* Call MLID */
        IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        DIOBoard = 0;
    }

    /* *****
    * DIOGetSN(char *serialNum)
    *
    * Description:
    * This routine gets the serial number from the adapter.
    * It is used for explicit requests of packages that are
    * for sale.
    *
    * Input:
    * serialNum - Pointer to where to store seri
    * al number
    *
    * Output:
    * serialNum - filled out if successful
    *
    * Returns:
    * 0 if successful
    * *****
    LONG DIOGetSN(char *serialNum)
    {
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 108)};
        if (!DIOBoard)
            return(-1);
    }
    */

```

```

ecb.ECB_StackID = MLID_GET_SN;
ecb.ECB_Fragment[0].FragmentAddress = serialNum;

```

```

/* Call MLID */
ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
return(ccode);

```

```

/*****
 *
 * DIOSignText(char *textToSign,
 *             LONG textLength,
 *             char *signature)
 *
 *
 * Description:
 * This routine uses the adapter to calculate a signature
 * for the given text.
 * It is used for explicit requests of packages that are
 * for sale.
 *
 * Input:
 * textToSign
 * textLength
 * signature
 *
 * Output:
 * signature
 *
 * Returns:
 * 0 if successful
 *
 * *****/

```

```

Description:
This routine uses the adapter to calculate a signature
for the given text.
It is used for explicit requests of packages that are
for sale.

```

```

Input:
textToSign
textLength
signature
- string to be signed
- length of string to be signed
- string to store signature

```

```

Output:
signature
- filled out if successful

```

```

Returns:
0 if successful

```

```

/*****
 *
 * DIOSignText(char *textToSign,
 *             LONG textLength,
 *             char *signature)
 *
 *
 * Description:
 * This routine uses the adapter to calculate a signature
 * for the given text.
 * It is used for explicit requests of packages that are
 * for sale.
 *
 * Input:
 * textToSign
 * textLength
 * signature
 *
 * Output:
 * signature
 *
 * Returns:
 * 0 if successful
 *
 * *****/

```

```

LONG DIOSignText(char *textToSign,
                 LONG textLength,
                 char *signature)
{

```

```

    LONG ccode;
    LONG fragment[3];
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 139)};

```

```

    if (!DIOBoard)
        return(-1);

```

```

    fragment[0] = (LONG)textToSign;
    fragment[1] = textLength;
    fragment[2] = (LONG)signature;

```

```

    ecb.ECB_StackID = MLID_SIGN_TEXT;
    ecb.ECB_Fragment[0].FragmentAddress = fragment;

```

```

    /* Call MLID */
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

```

```

/*****
 *
 * EXPORTED FUNCTION
 *
 * DPCGetMLIDStats(struct DriverStatsStructure **stats)
 *
 *
 * Description:
 * This routine fills in a pointer to the MLID stats
 * table.
 *
 * Input:
 * stats
 * Pointer to where to store poin
 *
 * Output:
 * stats filled in if successful
 *
 * Returns:
 * 0 if MLID is active
 *
 * *****/

```

```

EXPORTED FUNCTION

```

```

DPCGetMLIDStats(struct DriverStatsStructure **stats)
{

```

```

Description:
This routine fills in a pointer to the MLID stats
table.

```

```

Input:
stats
Pointer to stats table

```

```

Output:
stats filled in if successful

```

```

Returns:
0 if MLID is active

```

```

/*****
 *
 * DPCGetMLIDStats(struct DriverStatsStructure **stats)
 *
 *
 * Description:
 * This routine fills in a pointer to the MLID stats
 * table.
 *
 * Input:
 * stats
 * Pointer to stats table
 *
 * Output:
 * stats filled in if successful
 *
 * Returns:
 * 0 if MLID is active
 *
 * *****/

```

```

int DPCGetMLIDStats(struct DriverStatsStructure **stats)
{
    if (!DIOBoard)
        return(-1);

```

```

    *stats = DIOStats = (struct DriverStatsStructure *)
        CommandMlid(DIOBoard, 1, DIOControlEntry);

```

```

    return(0);
}

```

```

int DIOGetMLIDConfig(struct DriverConfigurationStructure **config)
{

```

```

    if (!DIOBoard)
        return(-1);

```

```

    *config = (struct DriverConfigurationStructure *)
        CommandMlid(DIOBoard, 0, DIOControlEntry);

```

```

    return(0);
}

```

```

/*****
 *
 * DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
 *
 *
 * Description:
 * This routine attempts to open an adapter channel for
 * the passed in bypass address, such as 0f 00 00 00 00.
 *
 * Input:
 * address
 * address for open
 * esr
 * - ESR address for this channel
 * channel
 * channel number is returned
 *
 * Output:
 * channel is filled out if successful
 *
 * Returns:
 * 0 if channel was opened
 *
 * *****/

```

```

DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
{

```

```

    Description:
    This routine attempts to open an adapter channel for
    the passed in bypass address, such as 0f 00 00 00 00.

```

```

    Input:
    address
    address for open
    esr
    - ESR address for this channel
    channel
    channel number is returned

```

```

    Output:
    channel is filled out if successful

```

```

Returns:
0 if channel was opened

```

```

/*****
 *
 * DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
 *
 *
 * Description:
 * This routine attempts to open an adapter channel for
 * the passed in bypass address, such as 0f 00 00 00 00.
 *
 * Input:
 * address
 * address for open
 * esr
 * - ESR address for this channel
 * channel
 * channel number is returned
 *
 * Output:
 * channel is filled out if successful
 *
 * Returns:
 * 0 if channel was opened
 *
 * *****/

```

```

LONG DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
{
    ChannelConfig cfg;

```

```

LONG
ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 142)};

if (!DIOBoard) {
    if (DIORegisterWithAdapter(DPCName))
        return(-1);
}

/* Initialize channel to 0. MLID will overwrite this */
cfg.CfgChannel = 0;

/* Point ESR to our packet handler */
cfg.CfgESR = esr;

/* Number of addresses to add */
cfg.CfgNumAddresses = 1;

/* Address of 0f 00 00 00 00 */
CMovB(address, cfg.CfgAddress, 6);

ecb.ECB_StackID = MLID_OPEN_CHANNEL;
ecb.ECB_Fragment[0].FragmentAddress = &cfg;

/* Call MLID */
ccode = IoctlMLID(DIOBoard, &ecb, DIOControlEntry);
if (ccode == 0)
{
    /* Store channel for close channel, add addr and delete addr */
    *channel = cfg.CfgChannel;
}
return(ccode);
}

/*****
 *
 * DIOCloseChannel (LONG channel)
 *
 * Description: This routine closes a previously opened channel.
 *
 * Input: channel - channel
 *
 * Output: nothing
 *
 * Returns: 0 if channel was closed
 *
 *****/
LONG
DIOCloseChannel (LONG channel)
{
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 157)};
    if (!DIOBoard)
        return(-1);

    ecb.ECB_StackID = MLID_CLOSE_CHANNEL;
    ecb.ECB_Fragment[0].FragmentAddress = &channel;
    ccode = IoctlMLID(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

/*****
 *
 * DIODelAddress (LONG channel, BYTE *address)
 *
 * ChannelConfig channelCfg;
 * LONG ccode;
 * ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 180)};
 * if (!DIOBoard)
 *     return(-1);
 *
 * channelCfg.CfgChannel = channel;
 * channelCfg.CfgNumAddresses = 1;
 * CMovB(address, channelCfg.CfgAddress, 6);
 *
 * ecb.ECB_StackID = MLID_DEL_ADDRESS;
 * ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
 * ccode = IoctlMLID(DIOBoard, &ecb, DIOControlEntry);
 * return(ccode);
 *
 *****/
LONG
DIODelAddress (LONG channel, BYTE *address)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 180)};
    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 6);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMLID(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

LONG
DIOAddGroupAddress (LONG channel, BYTE *address, BYTE *groupAddress)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 209)};
    BYTE key[8];

```



```

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 8);
    CMovB(groupAddress, key, 8);
    reverse_key((BYTE*)&key);
    CMovB(key, channelCfg.CfgGroupKey, 8);
    CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

int DIOAddHiAddr(unsigned char channel, BYTE *hiAddr)
{
    chunk key;
    ID hi_id;
    int i, ret = CAS_ERROR;
    ChannelConfig channelCfg;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 471)};

    if (!DIOBoard)
        return(-1);

    for(i = 0; i < 3; i++)
        hi_id[i] = 0x00;
    make_hi_key(&key);
    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;

    CMovB(hiAddr, channelCfg.CfgAddress, 8);
    /* Some strange things ... */
    reverse_key((BYTE*)&key);
    CMovB(&key, channelCfg.CfgGroupKey, 8);
    CMovB(&key, channelCfg.CfgElementKey, 8);

    channelCfg.CfgChannel = FDBChannel;
    channelCfg.CfgESR = FDB_ESR;
    channelCfg.CfgNumAddresses = 1;
    CMovB(&addr, channelCfg.CfgAddress, 8);
    CMovB(&pacau->g_key, key, 8);
    reverse_key(&key);
    CMovB(key, channelCfg.CfgGroupKey, 8);
    CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

    channelCfg.CfgESR = (int (*)())0xffffffff;
    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    CMovB(&addr, node->file_address, 6);

    ret = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);

    if((ret = WbioddAddAddress
        (BICDD_CHANNEL_CONFIG FAR *)&channel_config)) != CAS_OK)
        return ret;

    int DIORegisterSend(int (*sendRoutine)(TCB *))
{

```

```

    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 481)};

    if (!DIOControlEntry)
        return(-1);

    ecb.ECB_StackID = MLID_REGISTER_SEND_ROUTINE;
    ecb.ECB_Fragment[0].FragmentAddress = &sendRoutine;

    /* Call MLID */
    IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(0);
}

int DIOObtainReturnTCB(void (**returnTCBRoutine)(TCB *))
{
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 147)};

    if (!DIOControlEntry)
        return(-1);

    ecb.ECB_StackID = MLID_RETURN_TCB_ROUTINE;
    ecb.ECB_Fragment[0].FragmentAddress = returnTCBRoutine;

    /* Call MLID */
    IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(0);
}

int DPGGetIPAddress(LONG* ip) {
    LONG id;
    LONG (*ctl)(LONG board, ...);
    char buf[80];
    char* s = buf;

    if (CLSLGetStackIDFromName("\\002IP", &id))
        return 1;
    if (CLSLGetProtocolControlEntry(id, DIOBoard, &ctl))
        return 2;
    if (GetProtocolStringForBoard(ctl, DIOBoard, buf))
        return 3;
    s = strpbrk(buf, "0123456789");
    if (!s)
        return 4;
    *ip = inet_addr(s);
    if (*ip == (LONG)(-1))
        return 5;
    return 0;
}

```

```
#include "dpcagent.h"
#include <string.h>
#include <stdlib.h>
```

```
LONG DPCMaxConnections = 3;
int PackageDelivery = FALSE;
```

```
DlOcfgr_t DlOcfgr = {
    1330, // freq
    (198 << 24) | (77 << 16) | (117 << 8) | 21, // ip_address
    (198 << 24) | (77 << 16) | (117 << 8) | 66, // gateway_address
    1500, // mtu
    MSG("ATDT1-800-332-8071", 100), // tinet_phone_num
    MSG("ATDT1-800-825-3954", 187), // pdeliv_phone_num
    "", // dialout_prefix
    300, // tinet_inactivity_timer
    2, // pdeliv_inactivity_timer
    1, // modem_type
    120, // packet_lifetime - Max time to keep da
    ta in buffer
    60, // call_setup_timeout
    MSG("ATH0", 189), // hangup_str
    MSG("NO CARRIER", 190), // disconnect_str
    MSG("+++", 191), // escape_str
    MSG("CONNECT", 192), // connect_str
    MSG("ATE1Q0V1X4C1d2 S7=60 S11=55", 193), // init_str
    1024, // max_db_entries
    5, // tinet_baud_index(19200)
    0, // pdeliv_baud_index(2400)
    FALSE, // async_buffer_size(8K)
    "", // auto_login
    "", // wait_for_1.wait_for_9
    "", // send_1..send_9
    OUT_SLIP, // out_protocol
    TRUE, // obsolete(was tunnel)
    10, 5, 5, 5, 5, 5, 5, // wait_timeout_1..wait_timeout_9
    "", // ppp_login
    1400, // ppp_passwd
    0, // ppp_mru
    "", // ppp_accm
    "", // base_license
    "", // key
    -1, // net_interface
    {"0\0\0\0\0\0", // net_addr
    { "", "", "", "", "", "", "", // add_license
    }
    );
};
```

```
/*****
```

```
* EXPORTED FUNCTION
```

```
* DPCUpdateConfig(void)
```

```
* Description:
```

```
This routine opens \DIRECPC\DB\MODEM.CFG and updates our global
structures. If the file doesn't exist, or if its an older versio
(because its smaller), read in what you can and write out a new
one.
```

```
* Input:
```

```
* Output:
```

```
* void DPCUpdateConfigFile(void) {
```

```
0 if successful
-1 unable to open or create file
-2 unable to update file
```

```
*****
```

```
void ConfigSanityCheck(int handle)
```

```
{
    int changeFlag = 0;
    int i;
    LONG *timeout;

    if (DlOcfgr.wait_timeout_1 < 2 || DlOcfgr.wait_timeout_1 > 60)
    {
        changeFlag++;
        DlOcfgr.wait_timeout_1 = 10;
    }

    for (i = 0, timeout = &DlOcfgr.wait_timeout_2; i < 8; i++, timeout++)
    {
        if (*timeout < 2 || *timeout > 60)
        {
            changeFlag++;
            *timeout = 5;
        }

        if (changeFlag)
        {
            lseek(handle, 0, SEEK_SET);
            write(handle, &DlOcfgr, sizeof(DlOcfgr));
        }

        int ccode = -1;
        int handle;

        handle = open(MSG("SYS:DIRECPC\\DB\\MODEM.CFG", 194),
            O_RDWR | O_CREAT,
            S_IWRITE | S_IREAD);
        if (handle != -1)
        {
            if ( read(handle, &DlOcfgr, sizeof(DlOcfgr)) != sizeof(DlOcfgr))
            {
                lseek(handle, 0, SEEK_SET);
                if (write(handle, &DlOcfgr, sizeof(DlOcfgr)) != sizeof(DlOcfgr))
                {
                    ccode = -2;
                }
            }
            ConfigSanityCheck(handle);
            close(handle);
            ccode = 0;
        }
        DlOcfgr.ip_address = htonl(DlOcfgr.ip_address);
        DlOcfgr.gateway_address = htonl(DlOcfgr.gateway_address);
        return(ccode);
    }
}
```

```
void DPCUpdateConfigFile(void) {
```

```

int handle;

DloCfg.ip_address = htonl(DloCfg.ip_address);
DloCfg.gateway_address = htonl(DloCfg.gateway_address);
handle = open(MSG("SYS:DIRECTPC\\DB\\MODEM.CFG", 335),
              O_RDWR | O_CREAT,
              S_IWWRITE | S_IREAD);
if (handle != -1)
{
    write(handle, &DloCfg, sizeof(DloCfg));
    close(handle);
}
DloCfg.ip_address = htonl(DloCfg.ip_address);
DloCfg.gateway_address = htonl(DloCfg.gateway_address);
}

#define SerialWarn(s) sprintf(warnbuf, "\r\nDPCN: detected a bad serial number: %8.8s\r\n", (char*)(s)), ConsolePrintf(warnbuf), RingTheBell()

static inline unsigned long find_and_clear_low_bit(unsigned long* val) {
    unsigned long low = *val;
    if (low == 0)
        return 0;
    *val &= low - 1;
    return (*val ^ low);
}

static inline int parity(LONG serial) {
    int i;
    for (i = 0; find_and_clear_low_bit(&serial); ++i)
        return (i & 1);
}

static inline int UserCount(BYTE* s) {
    LONG serial = 0;
    int shift;

    s += 3;
    for (shift = 16; shift >= 0; shift -= 4) {
        serial |= (*s - ((*s >= 'A') ? 56 : 0x30)) << shift;
        ++s;
    }
    return 5 * ((serial & 0x00002) >> 1) |
        ((serial & 0x20000) >> 16) |
        ((serial & 0x02000) >> 11) |
        ((serial & 0x00040) >> 3));
}

void DPCSetMaxConnections(LONG* sum) {
    char warnbuf[120];
    int users;
    int pd = 0;
    int i;
    sum[0] = sum[1] = (-1);

    /* re-read modem.cfg file */
    i = DloCfg.ip_address;
    DPCUpdateConfig();
    DloCfg.ip_address = i;
}

if (strcmp(DloCfg.base_license, "Helius, Inc.", 8) == ESUCCESS) {
    DPCMaxConnections = 108;
    PackageDelivery = 1;
    memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
    return;
}

/* check for old license info */
if (atoi(DloCfg.base_license) < 02) {
    sprintf(warnbuf,
        "\r\nDPCN: deactivated old serial number: %8.8s\r\n",
        DloCfg.base_license);
    ConsolePrintf(warnbuf);
    RingTheBell();
    return;
}

/* handle base license first */
memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
if (parity(sum[0]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
if (parity(sum[1]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
users = UserCount(DloCfg.base_license);
if (DloCfg.base_license[2] == '*')
    pd = 1;

/* now handle additive licenses */
for (i = 0; i < 9; ++i) {
    int j;
    LONG serial[2];
    if (DloCfg.add_license[i][0] == 0)
        continue;
    /* check for duplicate license */
    for (j = i - 1; j >= 0; --j) {
        if (memcmp(DloCfg.add_license[i],
            DloCfg.add_license[j],
            sizeof(DloCfg.add_license[i])) == ESUCCESS) {
            memset(DloCfg.add_license[i], 0, sizeof(DloCfg.add_license[i]));
            DPCUpdateConfigFile();
            sprintf(warnbuf,
                "\r\nDPCN: deleted duplicate license number: %8.8s\r\n",
                DloCfg.add_license[j]);
            ConsolePrintf(warnbuf);
            RingTheBell();
            goto nextLicense;
        }
    }
    memcpy(serial, DloCfg.add_license[i], sizeof(serial));
    if (parity(serial[0]) == 1) {
        SerialWarn(DloCfg.add_license[i]);
        return;
    }
    if (parity(serial[1]) == 1) {
        SerialWarn(DloCfg.add_license[i]);
        return;
    }
    users += UserCount(DloCfg.add_license[i]);
    sum[0] += serial[0];
    sum[1] += serial[1];
    if (DloCfg.add_license[i][2] == '*')

```

Thu Jul 17 14:46:12 1997

license.c

Page 5

```
pd = 1;
nextLicense:
;
)
DPCMaxConnections = users * 4;
PackageDelivery = pd;
)
```



```

buffer[2],
buffer[3],
buffer[4],
buffer[5],
buffer[6],
buffer[7],
buffer[8],
buffer[9],
buffer[10],
buffer[11],
buffer[12],
buffer[13],
buffer[14],
buffer[15],
isprint(buffer[0]) ? buffer[0] : ' ',
isprint(buffer[1]) ? buffer[1] : ' ',
isprint(buffer[2]) ? buffer[2] : ' ',
isprint(buffer[3]) ? buffer[3] : ' ',
isprint(buffer[4]) ? buffer[4] : ' ',
isprint(buffer[5]) ? buffer[5] : ' ',
isprint(buffer[6]) ? buffer[6] : ' ',
isprint(buffer[7]) ? buffer[7] : ' ',
isprint(buffer[8]) ? buffer[8] : ' ',
isprint(buffer[9]) ? buffer[9] : ' ',
isprint(buffer[10]) ? buffer[10] : ' ',
isprint(buffer[11]) ? buffer[11] : ' ',
isprint(buffer[12]) ? buffer[12] : ' ',
isprint(buffer[13]) ? buffer[13] : ' ',
isprint(buffer[14]) ? buffer[14] : ' ',
isprint(buffer[15]) ? buffer[15] : ' ',
buffer += 16;
len -= 16;
}

if (len)
{
    /* the basic theory here is to build the buffer in place and the
       n insert the extra spaces */
    unsigned int n = 0;
    register char* d = display;

    while (n < len)
    {
        NWSprintf(d, MSG("%02x ", 483), buffer[n]);
        d += 3;
        display[(16 * 3 + 2) + n] = isprint(buffer[n]) ? buffer[
            ++n;
        ] : ' ';
        display[(16 * 3 + 2) + len] = 0;
        memset(d, ' ', (16 - len) * 3 + 2);
        d = display + (16 / 2 * 3);
        memmove(d + 2, d, sizeof(display) - (16 / 2 * 3) - 2);
        d[0] = d[1] = ' ';
        d = display + (16 * 3) + 4 + 8;
        memmove(d + 2, d, 9);
        d[0] = d[1] = ' ';
        puts(display);
    }
}

void DloUpdateModemStr( void )
{
    if (DloState == DLOS_IDLE)
        UpdateModemStr(MSG("Modem Status: IDLE
            \n", 201));
}

```

```

else if (DloState == DLOS_INIT)
    UpdateModemStr(MSG("Modem Status: Initializing Modem
        \n", 240));
else if (DloState == DLOS_DIAL)
    UpdateModemStr(MSG("Modem Status: Dialing
        \n", 236));
else if (DloState == DLOS_REDL)
    UpdateModemStr(MSG("Modem Status: Redialing
        \n", 235));
else if (DloState == DLOS_CONN)
{
    if (DloConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Connected to Package D
            \n", 241));
    else
    {
        if (ConnectBaudStr[0])
        {
            BYTE connectStr[80];

            NWSprintf(connectStr, MSG("Modem Status: Connect
                ed to Internet at%.24s\n", 473), ConnectBaudStr);
            UpdateModemStr(connectStr);
        }
        else
        {
            UpdateModemStr(MSG("Modem Status: Connected to I
                nternet
                \n", 184));
        }
    }
}

if (DloConn == DLO_CONN_PACKAGE)
    UpdateModemStr(MSG("Modem Status: Disconnecting from Pac
        kage Delivery
        \n", 101));
else
    UpdateModemStr(MSG("Modem Status: Disconnecting from Int
        ernet
        \n", 475));
}

int DloGetWriteBufferSize( void )
{
    LONG writeCount = 0;

    if (!AIOWriteBufferSize)
        return(2048);

    AIOGetPortStatus(AIOPortHandle, &writeCount, NULL, NULL, NULL, NUL
        L);
    return(AIOWriteBufferSize - writeCount);
}

int DloAndCommEmpty( void )
{
    if (DloConn == DLO_CONN_PACKAGE)
        return(DloPxmCount == 0);
    else
        return(DloIxmCount == 0);
}

void DloFlushReceive( void )
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
}

```

```

    DioRcvCount = 0;
    DioRcvIndex = 0;
    DioReadIndex = 0;
}

void DloStartConn(int timeout)
{
    if (DloNextConn == DLO_CONN_IDLE)
    {
        /* Assume package delivery connection first */
        DloNextConn = DLO_CONN_PACKAGE;
        if (timeout == DLO_PACKAGE_TIMEOUT)
        {
            DloInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9;
        }
        else if (timeout == DLO_INET_TIMEOUT)
        {
            DloInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
            DloNextConn = DLO_CONN_INET;
        }
        else if (timeout == DLO_GETKEYS_TIMEOUT)
        {
            DloInactivityTimer = 30 * 19;
        }
        else if (timeout > 2)
        {
            DloInactivityTimer = timeout * 19;
        }
        else
        {
            DloInactivityTimer = 2 * 19;
        }
    }

    if (DloState == DLOS_CONN && DloNextConn == DLO_CONN)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
        return;
    }

    if (DloConn == DLO_CONN_IDLE)
    {
        StateMachine(DLOE_SEND);
        DloConn = DloNextConn;
        DloNextConn = DLO_CONN_IDLE;
    }
    else if (DloConn == DLO_CONN_INET)
    {
        /* Package waiting for internet. Cause it to timeout quickly */
        DloStartTimer(19);
    }

    BaudRateHandler(int option, void *parameter)
    {
        parameter = parameter;
        return option;
    }

    static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
    {

```

```

    head = head;
    tail = tail;
    handle = handle;
    return;
}

```

```
void PDConfiguration(void)
```

```

{
    int i;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    int save;
    DloCfg_t tmpDloCfg;
    MFCNTROL *mfctl1;
    int baud;

```

```
CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));
```

```
NWSInitForm(NUTHandle);
```

```
NWSGetListSortFunction(NUTHandle, &oldSortFunction);
NWSSetListSortFunction(NUTHandle, NoSortHandler);
```

```

i = 0;
NWSAppendCommentField(i, 2, MSG("Package Phone
: ", 311), NUTH

```

```

andle);
NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg.pdeliv_phone_num
,
    dial_chars, F_NO_HELP, NUTHandle);

```

```

i++;
baud = tmpDloCfg.pdeliv_baud_index;
NWSAppendCommentField(i, 2, MSG("Package Baud

```

```

andle);
mfctl1 = NWSInitMenuField(InxMSG("Baud Rate", 314), 10, 40, BaudRateHand
ler, NUTHandle);

```

```

NWSAppendToMenuField(mfctl1, InxMSG("2400", 315), 0, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("3600", 316), 1, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("4800", 317), 2, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("7200", 318), 3, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("9600", 319), 4, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("19200", 320), 5, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("38400", 321), 6, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("57600", 322), 7, NUTHandle);
NWSAppendToMenuField(mfctl1, InxMSG("115200", 323), 8, NUTHandle);
NWSAppendMenuField(i, 28, NORMAL_FIELD, &baud, mfctl1, NULL, NUTHandle);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Package Inactivity(sec) : ", 324), NUTH
andle);
NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.pdeliv_inac
tivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Max Database Entries : ", 327), NUTH
andle);
NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.max_db_entr
ies, 1, 65535, F_NO_HELP, NUTHandle);

```

```

i++;

```

```

save = NWSEditPortalForm(InxMSG("Package Delivery Configuration Editor",
308),

```

```

12, 40,
/* center line, column
i, 76,

```

```

/* form height, width */
F_VERIFY, F_NO_HELP, /* Control flags, help message */
InxMSG("Save Changes?", 328), /* Confirm message, hand
NUTHandle);
le */

NWSSetListSortFunction(NUTHandle, oldSortFunction);
NWSDestroyForm(NUTHandle);
if (!save)
    return;
tmpDlocfg.pdeliv_baud_index = baud;
CMovB(&tmpDlocfg, &dlocfg, sizeof(Dlocfg_t));
DPCUpdateConfigFile();
)
LONG
(
    int i;
    Dlocfg_t *tmpDlocfg;
    LONG save;
    key = key;
    changed = changed;
    handle = handle;
    tmpDlocfg = (Dlocfg_t *)fp->customData;
    if (NWSPushList(NUTHandle) == 0)
        return K_SELECT;
    NWSInitForm(NUTHandle);
    i = 0;
    NWSAppendCommentField(i, 2, MSG("Authentication User Name: ", 516), NUTH
andle);
    NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlocfg->ppp_login,
printables, F_NO_HELP, NUTHandle);
    i++;
    NWSAppendCommentField(i, 2, MSG("Authentication Password: ", 578), NUTH
andle);
    NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlocfg->ppp_password,
printables, F_NO_HELP, NUTHandle);
    i++;
    NWSAppendCommentField(i, 2, MSG("Maximum Receive Unit : ", 579), NUTH
andle);
    NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDlocfg->ppp_mru, 6
4, 16384, F_NO_HELP, NUTHandle);
    i++;
    NWSAppendCommentField(i, 2, MSG("Asynch. Control Char Map: 0x", 580), NU
THandle);
    NWSAppendHexField(i, 30, NORMAL_FIELD, (int *)&tmpDlocfg->ppp_accm, 0, 0
xffffffff, F_NO_HELP, NUTHandle);
    i++;
    save = NWSeditPortalForm(InxMSG("PPP Configuration Editor", 510),
12, 40,
/* center line, column */
i, 78,

```



```

LONG net_addr[6];
void (*ControlEntryPoint)(void) = 0;
struct DriverConfigurationStructure* dvrCfgr = 0;
if (sscanf(hw_addr, "%2x-%2x-%2x-%2x-%2x-%2x",
    &net_addr[0],
    &net_addr[1],
    &net_addr[2],
    &net_addr[3],
    &net_addr[4],
    &net_addr[5]) != 6)
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Format error in Mac Address", 680));
    goto retry;
}
for (i = 0; i < 6; ++i)
    tmpDloCfgr->net_addr[i] = (BYTE)net_addr[i];
if (CLSLGetMLIDControlEntry(interface,
    &ControlEntryPoint))
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Interface not found", 681));
    goto retry;
}
dvrCfgr = (struct DriverConfigurationStructure *)
    CommandMlid(interface, 0, (LONG)ControlEntryPoint);
if (!dvrCfgr)
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Could not retrieve Interface Configurati
ion Table", 682));
    goto retry;
}
if ((dvrCfgr->DModeFlags & (1 << 6)) == 0)
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Interface does not support \"Raw Send\"
", 683));
    goto retry;
}
if (dvrCfgr->DMediaID != 2)
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Interface does not support ETHERNET_II
frames", 684));
    goto retry;
}
if (!CStrCmp(dvrCfgr->DShortName, DPCName))
{
    NWSAlert(12, 40, NUTHANDLE,
        InxMSG("Do not use DPC as the Interface", 685));
    goto retry;
}
tmpDloCfgr->net_interface = interface;
}

NWSDestroyForm(NUTHANDLE);
NWSPopList(NUTHANDLE);
return K_SELECT;
}

LONG ModifyLoginScript(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
    int i;

```

```

DloCfgr_t
LONG save;

```

```

key = key;
changed = changed;
handle = handle;

```

```

tmpDloCfgr = (DloCfgr_t *)fp->customData;
if (NWSPushList(NUTHANDLE) == 0)
    return K_SELECT;

```

```

NWSInitForm(NUTHANDLE);

```

```

i = 0;

```

```

NWSAppendCommentField(i, 2, MSG("Wait1: ", 581), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->wait_for_1,
    printables, F_NO_HELP, NUTHANDLE);

```

```

NWSAppendCommentField(i, 40, MSG("Wait Timeout 1: ", 599), NUTHANDLE);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfgr->wait_timeo
ut_1, 2, 60, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Send1: ", 518), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->send_1,
    printables, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Wait2: ", 520), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->wait_for_2,
    printables, F_NO_HELP, NUTHANDLE);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 2: ", 600), NUTHANDLE);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfgr->wait_timeo
ut_2, 2, 60, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Send2: ", 522), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->send_2,
    printables, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Wait3: ", 524), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->wait_for_3,
    printables, F_NO_HELP, NUTHANDLE);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 3: ", 601), NUTHANDLE);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfgr->wait_timeo
ut_3, 2, 60, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Send3: ", 526), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->send_3,
    printables, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Wait4: ", 528), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->wait_for_4,
    printables, F_NO_HELP, NUTHANDLE);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 4: ", 602), NUTHANDLE);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfgr->wait_timeo
ut_4, 2, 60, F_NO_HELP, NUTHANDLE);

```

```

i++;
NWSAppendCommentField(i, 2, MSG("Send4: ", 530), NUTHANDLE);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfgr->send_4,

```

```
printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait5: ", 532), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_5,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 5: ", 603), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *) &tmpDloCfg->wait_timeo
ut_5, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send5: ", 534), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_5,
    printables, F_NO_HELP, NUTHandle);
i++;
NWSAppendCommentField(i, 2, MSG("Wait6: ", 536), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_6,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 6: ", 604), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *) &tmpDloCfg->wait_timeo
ut_6, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send6: ", 538), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_6,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait7: ", 540), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_7,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 7: ", 605), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *) &tmpDloCfg->wait_timeo
ut_7, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send7: ", 542), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_7,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait8: ", 544), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_8,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 8: ", 606), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *) &tmpDloCfg->wait_timeo
ut_8, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send8: ", 546), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_8,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait9: ", 548), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_9,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 9: ", 607), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *) &tmpDloCfg->wait_timeo
ut_9, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send9: ", 550), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_9,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
save = NWSEditPortalForm(InxMSG("Login Script Editor", 582),
    12, 40, column, * /
/* center line, column
    1, 78,
/* form height, width
    F_NOVERIFY, F_NO_HELP, /* Control flags, help message */
    InxMSG("Save Changes?", 583),
    NUTHandle); /* Confirm message, hand
```

```
le */
//
// if (save)
// {
//
// CMovB(tmpDloCfg->wait_for_1, DloCfg.wait_for_1, 30*18);
// CMovB(&tmpDloCfg->wait_timeout_1, &dloCfg.wait_timeout_1, 9 * si
// zeof(LONG));
//
// DPCUpdateConfigFile();
//
// NWSDestroyForm(NUTHandle);
// NWSPopList(NUTHandle);
// return K_SELECT;
//
// }
```

```
int NewProtocolFlag = -1;
LONG ChangeProtocol(FIELD *fp, int key, int *changed, NUTInfo *handle)
```

```
{
    DloCfg_t *tmpDloCfg;
    LIST *listPtr, *slipList, *pppList, *netList;
    LONG ccode;
    LONG rcode = K_SELECT;
```

```
key = key;
changed = changed;
handle = handle;
```

```
tmpDloCfg = (DloCfg_t *)fp->customData;
```

```
if (NWSPushList(NUTHandle) == 0)
    return rcode;
```

```
NWSInitList(NUTHandle, NULL);
```

```
slipList = NWSAppendToList(MSG("Modem - SLIP", 519), (void *)OUT_SLIP, N
    UTHandle);
```

```
pppList = NWSAppendToList(MSG("Modem - PPP", 521), (void *)OUT_PPP, NUTH
    andle);
```

```
netList = NWSAppendToList(MSG("LAN/WAN", 537), (void *)OUT_NETWORK, NUTH
    andle);
```

```
if (tmpDloCfg->out_protocol == OUT_SLIP)
```

```
{
    listPtr = slipList;
```

```
else if (tmpDloCfg->out_protocol == OUT_PPP)
```

```
{
    listPtr = pppList;
```

```
else
```

```
{
    listPtr = netList;
```

```
ccode = NWSList(
```

```

InxMSG("Outbound Protocol", 509),
12, 40,
3,
16,
M_ESCAPE | M_SELECT,
&listPtr,
NUTHandle, NULL,
NULL, NULL);

if (ccode == M_SELECT)
{
    tmpDloCfg->out_protocol = NewProtocolFlag = (int)listPtr->otherI
        rcode = K_ESCAPE;
}

NWSDestroyList(NUTHandle);
NWSPopList(NUTHandle);
return rcode;
}

//LONG ChangeTunnel(FIELD *fp, int key, int *changed, NUTInfo *handle)
//{
//    DloCfg_t *tmpDloCfg;
//    LIST *listPtr, *enableList, *disableList;
//    LONG ccode;
//    LONG rcode = K_SELECT;
//
//    key = key;
//    changed = changed;
//    handle = handle;
//
//    tmpDloCfg = (DloCfg_t *)fp->customData;
//
//    if (NWSPushList(NUTHandle) == 0)
//        return rcode;
//
//    NWSInitList(NUTHandle, NULL);
//
//    enableList = NWSAppendToList(MSG("Enabled", 624), (void *)1, NUTHandle);
//    disableList = NWSAppendToList(MSG("Disabled", 625), (void *)0, NUTHandle
//);
//
//    if (tmpDloCfg->tunnel)
//    {
//        listPtr = enableList;
//    }
//    else
//    {
//        listPtr = disableList;
//    }
//
//    ccode = NWSList(
//        InxMSG("Tunnel Header", 626),
//        12, 40,
//        2,
//        16,
//        M_ESCAPE | M_SELECT,
//        &listPtr,
//        NUTHandle, NULL,
//        NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDloCfg->tunnel = NewProtocolFlag = (int)listPtr->otherInfo;
//        rcode = K_ESCAPE;
//    }
//
//    InxMSG("Outbound Protocol", 509),
//    12, 40,
//    3,
//    16,
//    M_ESCAPE | M_SELECT,
//    &listPtr,
//    NUTHandle, NULL,
//    NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDloCfg->out_protocol = NewProtocolFlag = (int)listPtr->otherI
//        rcode = K_ESCAPE;
//    }
//
//    NWSDestroyList(NUTHandle);
//    NWSPopList(NUTHandle);
//    return rcode;
//}

void ProviderConfiguration(void)
{
    int i;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    int save; /* tmpDloCfg;
    DloCfg_t tmpDloCfg;
    int ip0, ip1, ip2, ip3;
    int gw0, gw1, gw2, gw3;
    MFCNTROL *mfctl0;
    int baud;
    FIELD *fp;
    char *protocolStr;
    char *pppConfigStr;
    int cflags = F_VERIFYF;
    char *tunnelStr;

    CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

    NewProtocolLoop:
    NewProtocolFlag = -1;
    NWSInitForm(NUTHandle);
    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 30, MSG("Outbound Protocol : ", 525), NUTHandle
);

    if (tmpDloCfg.out_protocol == OUT_SLIP)
        protocolStr = MSG("Modem - SLIP", 527);
    else if (tmpDloCfg.out_protocol == OUT_PPP)
        protocolStr = MSG("Modem - PPP", 529);
    else
        protocolStr = MSG("LAN/WAN", 584);

    fp = NWSAppendHotSpotField(i, 50, NORMAL_FIELD, protocolStr,
        ChangeProtocol, NUTHandle);
    fp->customData = &tmpDloCfg;

    i++;
    NWSAppendCommentField(i, 2, MSG("Internet Phone
    : ", 126), NU
    THandle);
    NWSAppendStringField(i, 30, 30, NORMAL_FIELD, tmpDloCfg.tinet_phone_num,
        dial_chars, F_NO_HELP, NUTHandle);

    i++;
    baud = tmpDloCfg.tinet_baud_index;
    NWSAppendCommentField(i, 2, MSG("Internet Baud
    : ", 128), NU
    THandle);
    mfctl0 = NWSInitMenuField(InxMSG("Baud Rate", 129), 10, 40, BaudRateHand
    ler, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("2400", 130), 0, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("3600", 131), 1, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("4800", 132), 2, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("7200", 133), 3, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("9600", 146), 4, NUTHandle);
}

```

```

NWSAppendMenuItemField(mfct10, InxMSG("19200", 297), 5, NUTHandle);
NWSAppendMenuItemField(mfct10, InxMSG("38400", 300), 6, NUTHandle);
NWSAppendMenuItemField(mfct10, InxMSG("57600", 304), 7, NUTHandle);
NWSAppendMenuItemField(mfct10, InxMSG("115200", 305), 8, NUTHandle);
NWSAppendMenuItemField(i, 30, NORMAL_FIELD, &baud, mfct10, NULL, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("Internet MTU : ", 309), NUTHandle);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.mtu, 1, 65535, F_NO_HELP, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("Internet Inactivity(sec) : ", 310), NUTHandle);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.tinet_inactivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("IP to IP tunneling : ", 537), NUTHandle);
NWSAppendBooleanField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDloCfg.tunnel, F_NO_HELP, NUTHandle);
if (tmpDloCfg.tunnel)
    tunnelStr = MSG("Enabled", 627);
else
    tunnelStr = MSG("Disabled", 628);

fp = NWSAppendHotSpotField(i, 30, NORMAL_FIELD, tunnelStr, ChangeTunnel, NUTHandle);
fp->customData = &tmpDloCfg;
if (tmpDloCfg.tunnel)
{
    LONG ip_address = ntohl(tmpDloCfg.ip_address);
    LONG gateway_address = ntohl(tmpDloCfg.gateway_address);
    i++;
    ip0 = (ip_address & 0xff);
    ip1 = (ip_address >> 8) & 0xff;
    ip2 = (ip_address >> 16) & 0xff;
    ip3 = (ip_address >> 24) & 0xff;
    NWSAppendCommentField(i, 2, MSG("IP address (ISP : ", 306), NUTHandle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, &ip3, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 34, NORMAL_FIELD, &ip2, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 38, NORMAL_FIELD, &ip1, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 42, NORMAL_FIELD, &ip0, 1, 255, F_NO_HELP, NUTHandle);

    i++;
    gw0 = (gateway_address & 0xff);
    gw1 = (gateway_address >> 8) & 0xff;
    gw2 = (gateway_address >> 16) & 0xff;
    gw3 = (gateway_address >> 24) & 0xff;
    NWSAppendCommentField(i, 2, MSG("Hybrid Gateway Address : ", 307), NUTHandle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, &gw3, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 34, NORMAL_FIELD, &gw2, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 38, NORMAL_FIELD, &gw1, 1, 255, F_NO_HELP, NUTHandle);
    NWSAppendIntegerField(i, 42, NORMAL_FIELD, &gw0, 1, 255, F_NO_HELP, NUTHandle);

```

```

LP, NUTHandle);

i++;
NWSAppendCommentField(i, 30, MSG("slip Configuration", 552), NUTHandle);
NWSAppendCommentField(i, 2, MSG("Auto Login Enabled : ", 553), NUTHandle);
NWSAppendBooleanField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDloCfg.auto_login, F_NO_HELP, NUTHandle);

i++;
protocolStr = MSG("Modify Auto Login Script", 535);
fp = NWSAppendHotSpotField(i, 25, NORMAL_FIELD, protocolStr, ModifyLoginScript, NUTHandle);
fp->customDataRelease = NWSFree;
fp->customData = &tmpDloCfg;

i++;
if (tmpDloCfg.out_protocol == OUT_PPP)
{
    pppConfigStr = MSG("Modify PPP Configuration", 608);
    fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD, pppConfigStr, ModifyPPPConfig, NUTHandle);
    fp->customData = &tmpDloCfg;
}
else if (tmpDloCfg.out_protocol == OUT_NETWORK)
{
    fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD, "Modify Network Configuration", ModifyNetConfig, NUTHandle);
    fp->customData = &tmpDloCfg;
}

i++;
/* save */ NWSeditPortalForm(InxMSG("Provider Configuration Editor", 555), 12, 40, /* center line, column */
i, 78, /* form height, width */
/* flags, help message */
/* cflags */ F_NOVERIFY, F_NO_HELP, /* Contr
ol flags, help message */
/* InxMSG("Save Changes?", 556) */ NULL, /* Confirm message, hand
NUTHandle);

NWSSetListSortFunction(NUTHandle, oldSortFunction);
NWSDestroyForm(NUTHandle);

if (NewProtocolFlag != -1)
{
    cflags = F_FORCE;
    goto NewProtocolLoop;
}

if (!save)
    return;

tmpDloCfg.ip_address = htonl((ip0) |
(ip1 << 8) |
(ip2 << 16) |

```

```

tmpDlocCfg.gateway_address = htonl((gw0) |
    (gw1 << 8) |
    (gw2 << 16) |
    (gw3 << 24));

tmpDlocCfg.tinet_baud_index = baud;

if (memcmp(&DlocCfg, &tmpDlocCfg, sizeof(DlocCfg)) == 0 ||
    NWSConfirm(InxMSG("Save Changes?", 612), 0, 0, TRUE, NULL,
        NUTHandle, NULL) == FALSE)
    return;

/*
 * Get newest wait_for and send strings in case ModifyLoginScript()
 * changed them.
 */

// CMovB(DlocCfg.wait_for_1, tmpDlocCfg.wait_for_1, 30 * 18);
// CMovB(&DlocCfg.wait_timeout_1, &tmpDlocCfg.wait_timeout_1, 9 * sizeof(LONG));

CMovB(&tmpDlocCfg, &DlocCfg, sizeof(DlocCfg_t));

InetChangeProtocol();

DPCUpdateConfigFile();

void ModemConfiguration(void)
{
    int i;
    int save;
    DlocCfg_t tmpDlocCfg;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

    CMovB(&DlocCfg, &tmpDlocCfg, sizeof(DlocCfg_t));

    NWSInitForm(NUTHandle);

    NWSGetListsSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListsSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 2, MSG("Packet Life(sec) : ", 325), NUTHandle);
    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.packet_life_time, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Call Setup(sec) : ", 326), NUTHandle);
    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.call_setup_timeout, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Async Buffer Size : ", 212), NUTHandle);
    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.async_buffer_size, 1500, 1024*100, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Dial Prefix : ", 329), NUTHandle);
    NWSAppendStringField(i, 24, 10, NORMAL_FIELD, tmpDlocCfg.dialout_prefix,

```

```

    modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Hangup Str : ", 330), NUTHandle);
    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.hangup_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Disconnect str : ", 331), NUTHandle);
    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.disconnect_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Escape Str : ", 332), NUTHandle);
    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.escape_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Connect str : ", 333), NUTHandle);
    NWSAppendStringField(i, 24, 50, NORMAL_FIELD, tmpDlocCfg.connect_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Init str : ", 334), NUTHandle);
    NWSAppendStringField(i, 24, 60, NORMAL_FIELD, tmpDlocCfg.init_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    save = NWSEditPortalForm(InxMSG("Modem Configuration Editor", 513),
        12, 40, * /
        i, 76,
        /* form height, width */
        F_VERIFY, F_NO_HELP, /* Control flags, help m
        message */
        InxMSG("Save Changes?", 514), /* Confirm message, hand
        NUTHandle);

    le */

    NWSSetListsSortFunction(NUTHandle, oldSortFunction);
    NWSDestroyForm(NUTHandle);

    if (!save)
        return;

    CMovB(&tmpDlocCfg, &DlocCfg, sizeof(DlocCfg_t));

    if (AIOPortHandle != -1)
    {
        AIOSetWriteBufferSize(AIOPortHandle, DlocCfg.async_buffer_size);
        AIOSetWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
        DlocMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
            BufferSize : DLOBUFSIZE;
        DlocMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
            BufferSize : DLOBUFSIZE;
    }

    DPCUpdateConfigFile();
}

```



```

*
* Description:
* This routine is called when the modem needs to be dialed.

```

```

* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing

```

```

*****

```

```

static void WriteCommPhoneNumber(void)

```

```

(
    int baudIndex, i;
    char *number;

    if (DloConn == DLO_CONN_PACKAGE)
        baudIndex = DloCfg.pdeliv_baud_index;
    else
        baudIndex = DloCfg.tinet_baud_index;

```

```

    AIOConfigurePort(AIOPortHandle,
        AIOBaudRateDefines[baudIndex],
        AIO_DATA_BITS_8, AIO_STOP_BITS_1,
        AIO_PARITY_NONE,
        AIO_SOFTWARE_FLOW_CONTROL_OFF | AIO_HARDWARE_FLOW_CONTROL

```

```

L_ON);

```

```

    if (DloCfg.dialout_prefix{0})
    {
        if (DloConn == DLO_CONN_PACKAGE)
            number = DloCfg.pdeliv_phone_num;
        else
            number = DloCfg.tinet_phone_num;
        for (i = 0; number[i]; i++)
        {
            if (number[i] < 'A' || number[i] > 'z')
                break;
            if (number[i] > 'Z' && number[i] < 'a')
                break;
        }
    }
    if (i)
        SendAIOData(number, i);

```

```

    /*
    SendAIOData(number, i);
    SendAIOData(DloCfg.dialout_prefix, CStrLen(DloCfg.dialout_prefix));
    SendAIOData(number[i], CStrLen(number[i]));
    */

```

```

    if (DloConn == DLO_CONN_PACKAGE)
        SendAIOData(DloCfg.pdeliv_phone_num, CStrLen(DloCfg.pdeliv_phone_num));
    else
        SendAIOData(DloCfg.tinet_phone_num, CStrLen(DloCfg.tinet_phone_num));

```

```

    SendAIOData(MSG("\r", 613), 1);
    if (DloState == DLOS_REDL)

```

```

UpdateModemStr(MSG("Modem Status: Redialing\n", 559));

```

```

else
    UpdateModemStr(MSG("Modem Status: Dialing\n", 560));

```

```

*****

```

```

ProcessDisconnect(void)

```

```

*
* Description:
* This routine is called when where attaching and we lose Carrier Detect.

```

```

* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing

```

```

*****

```

```

static void ProcessDisconnect(void)

```

```

(
    int count;

```

```

    if (DloConn == DLO_CONN_PACKAGE)
    {
        UpdateModemStr(MSG("Modem Status: Disconnected from Package Dell\n", 237));
        count = DloPxmmitCount;
    }
    else
    {

```

```

        UpdateModemStr(MSG("Modem Status: Disconnected from Internet\n", 472));
        count = DloIXmitCount;
    }

```

```

    if (count)
    {
        WriteCommPhoneNumber();
        DioStartTimer(DloCfg.call_setup_timeout * 19);
        DioState = DLOS_DIAL;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_DIAL);
    }
    else
    {

```

```

        DioStartTimer(1 * 19);
        DioState = DLOS_DISC_4;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_DISC_4);
    }

```

```

*****

```

```

DioEndConn(void)

```

```

*
* Description:

```

This routine is terminate a modem connection.

```

*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing

```

```

void DloEndConn(void)
{

```

```

    if (AIOPortHandle < 0)
        return;

```

```

    if (DloConn == DLO_CONN_PACKAGE)
        DloPxmCount = 0;

```

```

    else
        DloIXmitCount = 0;

```

```

    switch(DloState)
    {

```

```

        case DLOS_INIT:
        case DLOS_REDL:
        case DLOS_DIAL:
        case DLOS_CONN:

```

```

            AIOFlushBuffers(AIOPortHandle, (AIO_FLUSH_WRITE_BUFFER |
            AIO_FLUSH_READ_BUFFER));

```

```

            DloState = DLOS_CONN;
            StateMachine(DLOE_TIMEOUT);
            break;

```

```

        case DLOS_IDLE:
        case DLOS_DISC_1:
        case DLOS_DISC_2:
        case DLOS_DISC_3:
        case DLOS_DISC_4:
            StateMachine(DLOE_DISCONN);
            break;

```

```

    }

```

```

    _0003(void)
    {
        In IDLE state, got SND event

```

```

    Description:
        The state machine is in the IDLE state.
        We've received a SND request.

```

```

    Input:      nothing
    Output:     nothing
    Returns:    nothing

```

```

int DloConnected(void)
{

```

```

    LONG extStatus = 0, chgdExtStatus;

```

```

    if (AIOPortHandle < 0)
        return(FALSE);

```

```

    AIOGetExternalStatus(AIOPortHandle, &extStatus, &chgdExtStatus);
    if (extStatus & AIO_EXTSTA_DCD)
        return(TRUE);

```

```

    return(FALSE);

```

```

}

static void _0003(void)
{

```

```

    LONG extStatus, chgdExtStatus;

```

```

    InitializeAIO();
    if (AIOPortHandle < 0)
    {

```

```

        UpdateModemStr(MSG("Modem Status: ERROR - Unable to initialize A
        \n", 238));
        return;
    }

```

```

    if (AIOGetExternalStatus(AIOPortHandle, &extStatus, &chgdExtStatus)
        || !(extStatus & AIO_EXTSTA_DCD))
    {

```

```

        AIOFlushBuffers(AIOPortHandle,
            (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));

```

```

        SendAIOData(DloCfg.init_str, cstrlen(DloCfg.init_str));
        SendAIOData(MSG("\r", 614), 1);
        DloStartTimer(5 * 19);
        DloState = DLOS_INIT;

```

```

        if (DloConn == DLO_CONN_INET)

```

```

            inetStateChange(DLOS_INIT);
        UpdateModemStr(MSG("Modem Status: Initializing Modem
        \n", 561));
    }

```

```

    }
    else
    {

```

```

        if (DloConn == DLO_CONN_PACKAGE)
            DloStartTimer(DloInactivityTimer);

```

```

        else
            DloStartTimer(DloInactivityTimer);

```

```

        DloStartTimer(30*19);
        DloStopPacketLifeTimer();

```

```

        WriteCommXmitBuffer();

```

```

        DloState = DLOS_CONN;

```

```

        if (DloConn == DLO_CONN_INET)

```

```

            inetStateChange(DLOS_CONN);

```

```

        if (DloConn == DLO_CONN_PACKAGE)
            UpdateModemStr(MSG("Modem Status: Connected to Package D
            \n", 562));

```

```

    }
    else
    {

```

```

        UpdateModemStr(MSG("Modem Status: Connected to Internet
        \n", 473));

```

```

        if (ConnectBaudStr[0])
        {

```

```

            BYTE connectStr[80];

```

```

            Nwsprintf(connectStr, MSG("Modem Status: Connect
            ed to Internet at %.24s\n", 563), ConnectBaudStr);

```



```
UpdateModemStr(connectStr);
    }
    else
    {
        UpdateModemStr(MSG("Modem Status: Connected to I
\n", 564));
    }
}

/*
***** In INIT state, got TIMEOUT *****
*/
static void _0100(void)
{
    Description:
        The state machine is in the INIT state.
        We timed out waiting for the modem init sequence.

    Input:    nothing
    Output:   nothing
    Returns:  nothing
}
*****

static void _0100(void)
{
    LONG    extStatus, chgdExtStatus;

    if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
        || ! (extStatus & AIO_EXTSTA_DCD))
    {
        WriteCommPhoneNumber();
        DloStartTimer(DloCfg.call_setup_timeout * 19);
        DloState = DLOS_DIAL;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_DIAL);
    }
    else
    {
        UpdateModemStr(MSG("Modem Status: Error - Still Connected
\n", 242));
        DloEndConn();
    }
}
/*
***** In INIT state, got OK response from mode *****
*/
static void _0104(void)
{
    Description:
        The state machine is in the INIT state.
        We've received an OK response from sending modem init sequence.

    Input:    nothing
    Output:   nothing
}
*****
```

```
*****
static void _0104(void)
{
    WriteCommPhoneNumber();

    DloStartTimer(DloCfg.call_setup_timeout * 19);
    DloState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}
/*
***** In DIAL state, got TIMEOUT *****
*/
static void _0200(void)
{
    Description:
        The state machine is in the DIAL state.
        It timed out waiting for connect.

    Input:    nothing
    Output:   nothing
    Returns:  nothing
}
*****

static void _0200(void)
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
    SendAIOData(MSG("\r", 615), 1);
    DloStartTimer(10 * 18);
    DloState = DLOS_REDL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_REDL);
}
/*
***** In DIAL state, got CONNECT response from modem *****
*/
static void _0201(void)
{
    Description:
        The state machine is in the DIAL state.
        We've received a CONNECT response from sending ATDT sequence.

    Input:    nothing
    Output:   nothing
    Returns:  nothing
}
*****

static void _0201(void)
{
}
*****
```

```
(
    WriteCommXmitBuffer();
    if (DloConn == DLO_CONN_PACKAGE)
    {
        DloStartTimer(DloInactivityTimer);
        UpdateModemStr(MSG("Modem Status: Connected to Package Delivery
        \n", 244));
    }
    else
    {
        DloStartTimer(DloInactivityTimer);
        UpdateModemStr(MSG("Modem Status: Connected to Internet
        \n", 474));
        if (ConnectBaudStr[0])
        {
            BYTE connectStr[80];

            NWsprintf(connectStr, MSG("Modem Status: Connected to In
            ternet at%.24s\n", 491), ConnectBaudStr);
            UpdateModemStr(connectStr);
        }
        else
        {
            UpdateModemStr(MSG("Modem Status: Connected to Internet
            \n", 492));
        }
    }

    DloStopPacketLifetime();
    DloState = DLOS_CONN;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_CONN);
)

/*****
*
* _0202(void)          In DIAL state, got disconnected
*
* Description:
*   The state machine is in the DIAL state.
*   We got disconnected from remote side.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* dem
*
* *****/
static void _0202(void)
{
    DloEndConn();
}

/*****
*
* _0205(void)          In DIAL state, got BUSY response from mo
*
* dem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received a BUSY response from sending ATDT sequence.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0205(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
    \n", 246));
    DloStartTimer(5 * 19);
}

/*****
*
* _0207(void)          In DIAL state, got RING response from mo
*
* dem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received a RING response from sending modem ATDT sequence.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0207(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
    \n", 246));
    DloStartTimer(5 * 19);
}

/*****
*
* _0206(void)          In DIAL state, got NODIALTONE response f
*
* rom modem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received an NODIALTONE response from sending ATDT sequence
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0206(void)
{
    UpdateModemStr(MSG("Modem Status: Busy
    \n", 245));
}

/*****
*
* _0206(void)          In DIAL state, got NODIALTONE response f
*
* rom modem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received an NODIALTONE response from sending ATDT sequence
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0206(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
    \n", 246));
    DloStartTimer(5 * 19);
}

/*****
*
* _0207(void)          In DIAL state, got RING response from mo
*
* dem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received a RING response from sending modem ATDT sequence.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0207(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
    \n", 246));
    DloStartTimer(5 * 19);
}

/*****
*
* _0205(void)          In DIAL state, got BUSY response from mo
*
* dem
*
* Description:
*   The state machine is in the DIAL state.
*   We've received a BUSY response from sending ATDT sequence.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
static void _0205(void)
{
    UpdateModemStr(MSG("Modem Status: No Dialtone
    \n", 246));
    DloStartTimer(5 * 19);
}

/*****/
```

```
static void _0207(void)
{
    UpdateModemStr(MSG("Modem Status: Ringing!!!\n", 247));
}

/*****
 *
 * _0208(void)
 *
 * In DIAL state, got NO ANSWER response fr
 *
 * Description:
 * The state machine is in the DIAL state.
 * We've received a NO ANSWER response from sending ATDT sequence.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0208(void)
{
    UpdateModemStr(MSG("Modem Status: No ANSWER\n", 248));
}

/*****
 *
 * _0300(void)
 *
 * In REDIAL state, got TIMEOUT
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We timed out.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0300(void)
{
    UpdateModemStr(MSG("Modem Status: Timeout - Reinitializing the modem\n", 249));
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
    SendAIOData(DloCfg.init_str, CStrlen(DloCfg.init_str));
    SendAIOData(MSG("\r", 616), 1);
    DloStartTimer(5 * 19);
    DloState = DLOS_INIT;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_INIT);
}

/*****
 *
 * _0302(void)
 *
 * In REDIAL state, got Disconnected
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We got disconnected by the remote site.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0302(void)
{
    DloEndConn();
}

/*****
 *
 * _0104(void)
 *
 * In REDIAL state, got OK response from mo
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We've received an OK response from sending modem init sequence.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0304(void)
{
    WriteCommPhoneNumber();
    DloStartTimer(DloCfg.call_setup_timeout * 19);
    DloState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}

/*****
 *
 * _0400(void)
 *
 * In CONNECTED state, timed out
 *
 * Description:
 * State: The state machine is in the CONNECTED state.
 * Event: We timed out due to inactivity.
 * Action: Add a 1.5 sec pre-escape delay. DLOS_DISC_1 state.
 *
 * Input: nothing
 *
 * Output:
 *
 *****/
```

```

*      nothing
*
*      Returns:      nothing
*
* *****
static void _0400(void)
{
    DloStartTimer( (1 * 19) + 9); /* 1.5 second delay */
    DloState = DLOS_DISC_1;
    if (DloNextConn == DloConn)
        DloNextConn = DLO_CONN_IDLE;
    if (DloConn == DLO_CONN_INET) {
        UpdateModemStr(MSG("Modem Status: Disconnecting from Internet
\n", 566));
        InetStateChange(DLOS_DISC_1);
    }
    else if (DloConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Disconnecting from Package Delivery
\n", 565));
    else
        UpdateModemStr("Modem Status: Disconnecting
\n");
}

/*****
*
*      _0402(void)      In CONNECT state, got disconnected
*
*      Description:
*      State: The state machine is in the CONNECT state.
*      Event: We were disconnected by the remote site.
*      Action: If still have data to send:
*
*      1800...), DLOS_DIAL state.
*
*      OS_DISC_4 state.
*
*      Input:      nothing
*
*      Output:     nothing
*
*      Returns:    nothing
*
* *****
static void _0402(void)
{
    ProcessDisconnect();
}

/*****
*
*      _0403(void)      In CONNECTED state, got SEND request
*
*      Description:
*      State: The state machine is in the CONNECTED state.
*      Event: We've received a request to send data to the remote site
*
*      Action: Extend the inactivity timer.
*
* *****
static void _0500(void)
{
    AIOFlushBuffers(AIOPortHandle,
                    (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));
    if (DebugFlag)
        fputs(DloCfg.escape_str, stdout);
    SendAIOData(DloCfg.escape_str, CStrLen(DloCfg.escape_str));
    DloStartTimer(2 * 19);
}

```

```

*      Input:      nothing
*
*      Output:     nothing
*
*      Returns:    nothing
*
* *****
static void _0403(void)
{
    if (DloConn == DLO_CONN_PACKAGE)
        DloStartTimer(DloPinactivityTimer);
    else
        DloStartTimer(DloInactivityTimer);
}

```

```

/*****
*
*      _0500(void)      In Disconnect 1 state, got timed out
*
*      Description:
*      State: Disconnect 1 state
*
*      Two ways to get in:
*      1)

```

```

- Inactivity timer kicked
- Set 1.5 pre-escape timer
2)
- Inactivity timer kicked in
- Set 1.5 pre-escape timer(DLO
- Sent escape string w/2
- Sent hangup string w/1
- Timed out

```

```

Event: Intentional 1.5 or 10 second timeout.
Action: Send escape string(+++) set 2 second timer, DLOS_DISC_2
state.

```

```

*      Input:      nothing
*
*      Output:     nothing
*
*      Returns:    nothing
*
* *****
static void _0500(void)
{
    AIOFlushBuffers(AIOPortHandle,
                    (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));
    if (DebugFlag)
        fputs(DloCfg.escape_str, stdout);
    SendAIOData(DloCfg.escape_str, CStrLen(DloCfg.escape_str));
    DloStartTimer(2 * 19);
}

```



```

BYTE *xmitBuffer;

if (DloNextConn == DLO_CONN_IDLE)
{
    /* Assume package delivery connection first */
    DloNextConn = DLO_CONN_PACKAGE;

    if (timeout == DLO_PACKAGE_TIMEOUT)
    {
        DloPinactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
    }
    else if (timeout == DLO_INET_TIMEOUT ||
             timeout == DLO_INET_NO_TIMEOUT)
    {
        DloInactivityTimer = DloCfg.tinet_inactivity_timer * 19
        DloNextConn = DLO_CONN_INET;
    }
    else if (timeout == DLO_GETKEYS_TIMEOUT)
    {
        DloPinactivityTimer = 30 * 19;
    }
    else if (timeout > 2)
    {
        DloInactivityTimer = timeout * 19;
    }
    else
    {
        DloPinactivityTimer = 2 * 19;
    }
}

#if USE_AIO_DEADMAN
/*
 * Update AIO deadman timer to timeout + 5 seconds
 */
AIOSetExternalControl(AIOPortHandle, AIO_SET_DEADMAN_TIMER, timeout + 5)
#endif

if (size > DLOBUFSIZE || size < 1)
    return 0;

if (DloState == DLOS_CONN && DloNextConn == DloConn)
{
    if (timeout != DLO_INET_NO_TIMEOUT)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
    }
    UpdateModemLights(1, 0, 1);
    if (DebugFlag)
    {
        printf(MSG("Sending %d bytes:\n", 485), size);
        HexAsciiDump(buffer,
                      (DloConn == DLO_CONN_INET && size > 32) ? 3
                      : size);
    }
    SendAIOData(buffer, size);
    return size;
}

if (DloNextConn == DLO_CONN_PACKAGE)
{
    maxBufferSize = &DloMaxBufferSize;
    xmitCount = &DloPXmitCount;
    xmitBuffer = DloPXmitBuffer;
}
else
{
    maxBufferSize = &DloIMaxBufferSize;
    xmitCount = &DloIXmitCount;
    xmitBuffer = DloIXmitBuffer;
}

if (size >= *maxBufferSize - *xmitCount)
    return 0;
for (i = 0; i < size) && (*xmitCount < *maxBufferSize); i++, (*xmitCount)++)
{
    if (!*xmitCount)
        DloStartPacketLifetimer();
    xmitBuffer[*xmitCount] = buffer[i];
}

if (DloConn == DLO_CONN_IDLE)
{
    StateMachine(DLOE_SEND);
    DloConn = DloNextConn;
    DloNextConn = DLO_CONN_IDLE;
}
else if (DloConn == DLO_CONN_INET)
{
    /* Package waiting for internet. Cause it to timeout quickly */
    DloStartTimer(19);
    return size;
}

*****
WriteCommXmitBuffer(void)
*****
Description:
    This routine is called after the modem is connected to send the
    data stored in DloXmitBuffer to the modem.

Input:    nothing
Output:   nothing
Returns:  nothing

*****
static void WriteCommXmitBuffer( void )
{
    BYTE *xmitBuffer;
    LONG *xmitCount;

    if (DloConn == DLO_CONN_PACKAGE)
    {
        xmitBuffer = DloPXmitBuffer;
        xmitCount = &DloPXmitCount;
    }
}

```



```

DloCallBackIndex = 0;
DloCallBackEscape = 0;
DloCallBackStarted = 0;
return(0);
)
/*****
*
* ValidPacket(BYTE *buf_to_rx,
*             int *len_to_rx)
*
* Description:
*   This routine validates a response from the modem. It checks the
*   header
*   length, checksum, opcode and status.
*
* Input:
*   buf_to_rx      - pointer to the response message
*   len_to_rx      - pointer to the total length of the message
*
* Output:
*   len_to_rx      - changed to the real size of the message
*   the header
*
* Returns:
*   TRUE if packet is valid
*****/
int ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    LroEspkt_t *msg;
    WORD frameCrc = 0;
    WORD slipCrc = 0;
    WORD frameLen = 0;
    WORD crcVal = 0xffff;

    msg = (LroEspkt_t *)buf_to_rx;
    frameLen = (msg->length) & 0x7fff;

    if ((*len_to_rx - sizeof(WORD)) == frameLen)
    {
        CMovB(&buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc));
        slipCrc = calcrc(crcVal, buf_to_rx, frameLen);
        if (slipCrc == frameCrc)
        {
            return(1);
        }
    }
    return(0);
}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    unsigned short frameCrc=0; // CRC value contained in the frame
    unsigned short frameLen=0; // Length value contained in the frame
    unsigned short slipCrc=0; // CRC returned from calculation
    LroAsyncMsg_t *msg;

```

```

DacauResponse_t *d_msg;
int status = FALSE;

msg = (LroAsyncMsg_t *)buf_to_rx;
// extract the frame length contained in the first 2 bytes of the frame
frameLen = msg->header.length;
frameLen &= 0x7fff;

if ((*len_to_rx - sizeof(unsigned short)) == frameLen)
{
    // since sliplen includes CRC
    // extract the crc contained in the last 2 bytes of the frame
    frameCrc = msg->data[frameLen - LRO_ASYNC_HDR_SIZE];
    frameCrc += msg->data[frameLen - LRO_ASYNC_HDR_SIZE + 1] * 256;

    slipCrc = calcrc (INITCRC, (unsigned char *) buf_to_rx, frameLen);
    if (slipCrc == frameCrc)
    {
        d_msg = (DacauResponse_t *) (msg->data);
        if (d_msg->opcode == 1 && d_msg->status == 0)
        {
            *len_to_rx = frameLen - RETURN_POINT;
            status = TRUE;
        }
    }
    return status;
}

/*****
*
* DloCallBackRead(void)
*
* Description:
*   This routine reads as many bytes as it can from the modem
*   on behalf of the process that scheduled a receive thru
*   DloScheduleReceive(). It's called by the main DLO thread
*   as long as a call back is scheduled(DloCallBack != 0) and
*   the modem has sent the previous request. All Slip specific
*   characters are stripped and all Slip escape characters are
*   converted back to ASCII. If the end of the message is hit,
*   call the processes event routine with the message.
*
* Input:
*   nothing
*
* Output:
*   nothing
*
* Returns:
*   nothing
*****/
static void DloCallBackRead()
{
    BYTE value;
    for(;;)
    {
        if (DloReceive(&value, 1) == 0)
            break;
        if (DebugFlag)
            putchar(value);
    }
}

```

```

if (DioCallBackStarted == 0)
{
    if (value == END)
    {
        DioCallBackStarted = 1;
    }
    continue;
}

switch(value)
{
    /* if it's an END character then we're done with the pac
ket */
    case END:
        /* We're done, */
        if (DioCallBackType == EXPLICIT_RECEIVE)
        {
            if (ValidExplicitPacket(DioCallB
ackBuffer, &DioCallBackIndex))
            {
                DioCallBack(DioCallBackB
uffer + LRO_EXPLICIT_HDR_SIZE,
ackIndex, 0);
            }
        }
        else
        {
            DioCallBack = 0;
            return;
        }
        DioCallBackIndex = 0;
        DioCallBackStarted = 0;
        DioCallBackEscape = 0;
        break;
    }
    else
    {
        /* We're done */
        if (DioCallBackType == EXPLICIT_RECEIVE)
        {
            DioCallBack(DioCallBackBuffer +
LRO_EXPLICIT_HDR_SIZE,
DioCallBackIndex
, 1);
        }
        else
        {
            DioCallBack(DioCallBackBuffer +
RETURN_POINT + LRO_ASYNC_HDR_SIZE,
DioCallBackIndex
, 1);
        }
        DioCallBack = 0;
        return;
    }
    break;
}

LONG DPNextRegistrationCheck;

/*****
*
* DioMain(void *parm)
*
* Description:
* Main thread for modem handling.
* We first initialize the modem thru AIO. We then check the
* modem to see if we've received anything. If we did, we gather
* it until we get a carriage return. Then we check against expecte
* responses. If we get a expected response, we set the appropriate
* event in the state machine. We then check for packet life
* and state machine timeouts. Otherwise, we sleep.
*
* Input:
* parm
*
* Output:
* nothing
*
* Returns:
* nothing
*****/

```

```

*****
void DloMain(void *parm)
{
    LONG curticks, delta;
    LONG count, bytesRead;
    WORD state;
    BYTE value;
    int event, eventLen;
    char *pPtrAtBegin, *pPtrAtEnd;
    LONG extStatus;

    parm = parm;
    DloLastKnownTickCount = GetCurrentTime();

    while(!ExitingFlag)
    {
        delay(1000 / 6);

        count = 0;
        bytesRead = 0;
        if (AIOGetPortStatus(AIOPortHandle,
            &count, /* write count */
            0, /* write status */
            &bytesRead, /* read count */
            0, /* read status */
            &extStatus,
            0) == 0)
        {
            extStatus &= AIO_EXTSTA_DCD;
            if (extStatus != DloLastDCD)
            {
                if (!extStatus)
                    StateMachine(DLOE_DISCONN);
                DloLastDCD = extStatus;
            }
        }
        UpdateModemLights(count, bytesRead, DloLastDCD);
        if (DloState == DLOS_IDLE)
        {
            if (DloPXMitCount)
            {
                DloConn = DLO_CONN_PACKAGE;
                StateMachine(DLOE_SEND);
            }
            else if (DloIXmitCount)
            {
                DloConn = DLO_CONN_INET;
                StateMachine(DLOE_SEND);
            }
        }
        curticks = GetCurrentTime();
        if (curticks < DloLastKnownTickCount)
        {
            delta = curticks + (0xffffffff - DloLastKnownTickCount);
        }
        else
        {
            delta = curticks - DloLastKnownTickCount;
        }
        DloLastKnownTickCount = curticks;
    }
}

```

```

    if (DIOStats && curticks > DPCNextRegistrationCheck)
    {
        char buf[16];
        LONG hw;
        double key;
        CustVars* customPtr = (CustVars*)&DIOStats->CustomVaria
bleCount);
        LONG rxFreq = customPtr->CustomVariable[1] / 10;
        DPCSetMaxConnections((LONG*)&key);
        if (strncmp(DloCfg.base_license, "Helius, Inc.", 8) == 0)
        {
            DPCNextRegistrationCheck = (LONG)(-1);
            goto skipRegistrationCheck;
        }
        /* get hardware serial number */
        *buf = 0;
        DIOGetSN(buf);
        hw = strtoul(buf, 0, 10);
        if (hw == 0) {
            ConsolePrintf("\r\nDPCAgent: could not obtain hardware
            serial number of DPC card\n");
            goto disableDPCAgent;
        }
        /* compute the registration key */
        if (DebugFlag == 0x98) {
            printf("\rRegCheck: %e\n", key);
            HexAsciiDump((void*)&key, sizeof(key));
        }
        key *= hw + DPC_IP_Address;
        if (DebugFlag == 0x98) {
            printf("\rRegCheck: %e %d %08x\n",
                key, hw, DPC_IP_Address);
            HexAsciiDump((void*)&key, sizeof(key));
        }
        if (key == *(double*)&DloCfg.key)
            DPCNextRegistrationCheck = curticks + 131072; /*
            120 minutes */
        else
        {
            ConsolePrintf("\r\nDPCAgent: detected a bad regi
            stration key\n");
            disableDPCAgent;
            DPCMaxConnections = 3;
            RingTheBell();
            if (DPCNextRegistrationCheck) {
                DloCfg.gateway_address = 0;
                StateMachine(DLOE_TIMEOUT);
                DPCNextRegistrationCheck = curticks + 2185; /*
                2 minutes */
            }
            else
            {
                DPCNextRegistrationCheck = curticks + 131072;
            }
            /* 120 minutes */
            if ((DPCNextRegistrationCheck & 0xfffffffff0) == 0xfffffffff)
            {
                DPCNextRegistrationCheck += 17; /* wrap */
            }
        }
        skipRegistrationCheck;
        if (AIOPortHandle >= 0)
        {

```


Thu Jul 17 14:46:11 1997

dloc

Page 55

```
timeout flag set */
localBackIndex, 1);

if (delta >= DlocCallBackTimeout)
{
    /* Timeout!! Call routine with
    DlocCallBackWait = 0;
    DlocCallBack(DlocCallBackBuffer, D
    DlocCallBack = 0;

    else
        DlocCallBackTimeout -= delta;

    /* Read any available modem characters f
    DlocCallBackRead();

    )
    )
    if (AIOPortHandle >= 0)
        AIOReleasePort(AIOPortHandle);

    DPCModemPID = 0;
}
```


page ,130

```
*****
; BEGIN_MANUAL_ENTRY ( History, DPC/HISTORY )
;
; DirecPC Driver Code for NetWare 386.
; This driver must be loaded after MSM.NLM and ETHERNET.NLM.
;
; Written by:   DFS
; Date:        November, 1995
;
; *****
; History Log:
; version 0.0 - First demo version
; version 0.2 - (3-15-96) Put SignalQuality and RxFreq into stats table
; version 0.3 - (4-09-96) Fixed CloseChannel so that DPCAGENT.NLM
;                 can be properly reloaded.
; version 0.4 - (4-16-96) Added code to send turbo internet packets
;                 up to the ethernet instead of DPCAGENT
; version 0.5 - (4-24-96) Adjusted RPacketOffset 2 bytes into
;                 envelope in order to build 14 byte ethernet header
;                 over DPC's 12 byte header to turbo internet packets
; version 0.6 - (5-16-96) Completion of Phase 2
;                 Debug screen only active if -DEBUG on the command line
; version 1.00 - (6-11-96)
; version 1.01 - (6-17-96)
;                 Added check to end of DriverInit for locked adapter
;                 Added Agent Register call to DriverManagement so that
;                 we can call agent when we're being removed, allowing
;                 agent to go into idle state until we're loaded again.
; version 1.02 - (6-20-96)
;                 Added Freq= to command line.
; version 1.03 - (6-24-96)
;                 Fixed packet too big bug.
; version 1.04 - (6-25-96)
;                 Really Fixed packet too big bug.
; version 1.05 - (6-28-96)
;                 Enhanced dpcagent.nlm
; version 1.06 - (7-10-96)
;                 Reduced MLIDMaximumSize so Netscape wouldn't overflow
;                 when tunneling enabled.
; version 1.07 - (7-13-96)
;                 Stopped calling SendComplete from DriverSend
;                 Added DriverTCBComplete to be obtained from IOCTL
;                 Bumped TxFreeCount to 32
; version 1.09a - (8-28-96)
;                 Reduced max packet size if we couldn't hook bind event.
; version 1.20 - 2-12-96
;                 Returned ECB allocated by FastProcessGetRCB
;                 Changed call to DataLinkRxFrame to DPCRxFrame
;                 Changed copyright string
;
; END_MANUAL_ENTRY
; *****
;
; name      DPC
; title     DirecPC LAN Driver (HSM Version)
; subttl   -- Structures and Equate Values --
; page
;
; include driver.inc
; subttl   -- DirecPC Specific Equates --
```

page

```
*****
; *****
; Equates.
; *****
; subttl   -- DirecPC Specific Structures --
; page
;
; MLID_MAJOR_VERSION      equ      01
; MLID_MINOR_VERSION      equ      20
;
; TRUE                    equ      01
; FALSE                   equ      00
;
; TIMESTAMP               equ      FALSE
; TIMESTAMP_BUFFER_SIZE   equ      (256 * 4)
;
; INT_3                   equ      int      3
;
; EVENT_PROTOCOL_BIND     equ      33
; EVENT_PROTOCOL_UNBIND   equ      34
;
; ; Status Register bit masks
;
; STAT_BUSY               equ      01
; STAT_ERROR              equ      02
;
; STAT_RX_INT             equ      04h
; STAT_NO_RBD             equ      08h
;
; STAT_CNTL_INT           equ      10h
; STAT_SOUTPUT            equ      20h
;
; ; Control Register bit masks
;
; CNTL_CPU_EN             equ      01h
; CNTL_RX_EN              equ      02h
; CNTL_INT_EN             equ      04h
; CNTL_CHAN_ATT           equ      08h
; CNTL_SINGL_INT          equ      10h
; CNTL_15_INT             equ      F0h
; CNTL_AUTO_INC           equ      100h
; CNTL_MRESET             equ      200h
; CNTL_FORCEINT           equ      400h
; CNTL_SOUTPUT            equ      800h
; CNTL_IRQ3               equ      0000h
; CNTL_IRQ4               equ      2000h
; CNTL_IRQ5               equ      4000h
; CNTL_IRQ9               equ      6000h
; CNTL_IRQ10              equ      8000h
; CNTL_IRQ11              equ      0A000h
; CNTL_IRQ12              equ      0C000h
; CNTL_IRQ15              equ      0E000h
;
; ; Synthesizer chip types
;
; INVALID_TUNER           equ      Offh
; SHARP                   equ      00h
; ALPS                    equ      08h
; PANASONIC               equ      20h
; SHARP_CUSTOM            equ      28h
;
; ; Define State Numbers (informational only)
```

```

;
INIT equ 0
SYNTH_PRGM equ 1
ACQ_PD_DELAY equ 2
ACQ_PD equ 3
ENABLE_BTR equ 4
START_SEARCH_FOR_FEC equ 5
CHECK_FOR_FEC_LOCK equ 6
SET_OTHER_MODE equ 7
TRACKING equ 8
POINTING_ACQ equ 9
POINTING_TRACKING equ 10
HALT equ 11
;
; New Stuff DBS
; demod command definitions
;
ACQUIRE_MODE equ 0
HALT_MODE equ 2
BUSY_MODE equ 3
POINTING_MODE equ 4
;
; start a new acquisition
; Do nothing
; Trying to acquire
; Special test mode
;
DEFAULT_RX_FREQ equ 1330
BIT_OFF equ 00h
; BtrControlAddr bits - write register 0
;
FREQ_PWR_MASK equ 080h
PHASE_PWR_MASK equ 07h
BTR_SENSE_MASK equ 08h
BTR_ERR_ENA_MASK equ 10h
FREQ_PWR_OFFSET equ 20h
PHASE_PWR_OFFSET equ 01h
; AfcControlAddr bits - write register 1
;
SWP_ENA_MASK equ 01h
SWEEP_DIR_SENSE_MASK equ 08h
AFC_SENSE_MASK equ 10h
EXT_INT_MASK equ 20h
BPSK_MASK equ 40h
ROM_ENA_MASK equ 80h
SQF_PEAK_EN_MASK equ 02h
; BitDetControlAddr bits - write register 2
;
SOFT_THRS_MASK equ 1Fh
DECODER_INFC_SEL_MASK equ 80h
VIT_SEQ_MASK equ 40h
SOFT_THRS_OFFSET equ 01h
; AgcFirControlAddr bits - write register 3
;
AGC_REF_MASK equ 1Fh
AGC_SENSE_MASK equ 40h
FIR_BYPASS_MASK equ 80h
SWAP_IQ_MASK equ 20h
AGC_REF_OFFSET equ 01h
;
; CrlkControlAddr bits - write register B
;
CRLK_DET_PWR_MASK equ 07h
CRLK_FC_MASK equ 38h
CRLK_GAIN_MASK equ C0h
;
CRLK_DET_PWR_OFFSET equ 01h
CRLK_FC_OFFSET equ 08h
CRLK_GAIN_OFFSET equ 40h
;
; SynthSerControlAddr:bits - write register C
;
SDATA_MASK equ 01h
SCLK_MASK equ 02h
SENA_MASK equ 04h
MODE_MASK equ 08h
CRL_ACC_ENABLE equ 10h
DEPUNC_BYPASS_MASK equ 20h
RESET_FEC_ACQ_MASK equ 40h
RESET_BTR_ACC_MASK equ 80h
;
; DaadOffsetControlAddr bits - write register E
;
I_CHANNEL_OFFSET equ 01h
CRL_ERROR_OFFSET equ 08h
BTR_ERROR_OFFSET equ 40h
;
;
SYNTH_LOCK_MASK equ 01h
CRL_LOCK_MASK equ 02h
SWEEPING_MASK equ 04h
DIR_MASK equ 08h
FEC_LOCK_MASK equ 10h
TUNER_TYPE_MASK equ 20h
TUNER_ID_MASK equ 0f0h
;
; Frequency offsets */
;
PLUS_OFFSET equ 40
ZERO_OFFSET equ 0
MINUS_OFFSET equ -40
OFFSET_THRESHOLD equ 1152
FREQ_BASE equ 9011
K_TRACK equ 1736
K_REACQ equ 29622
NOM_COUNT_TRACK equ 48761
NOM_COUNT_REACQ equ 19432
SYNTH_CHANNEL_SIZE equ 3645
SYNTH_RATIO equ 64
SYNTH_CHANNELS_PER_STEP equ 40
SYNTH_FIRST_CHANNEL equ 3788
;
; BPSK equ BPSK_MASK OR ROM_ENA_MASK
;
; Possible Viterbi modes */
;
LOWRATE equ 0
HIGHRATE equ 1
;
; demod status states */
;
UNLOCKED equ 0
LOCKED equ 1
;
; Configuration equates
;
RBD_BASE_ADDR equ 0a000h
ADAP_RBD_NUM equ 128
RBD_BUFFER_SIZE equ 1024
LOCAL_BUF_NUM equ 400

```

```
; RBD status bits.
;
; FRAMING_ERR      equ 0001h      ; Framing error
; CRC_ERR          equ 0002h      ; CRC error
; ABORT             equ 0004h      ; Abort
; ALIGN_ERR        equ 0008h      ; Alignment error
; DES_ERR          equ 0010h      ; DES Error
; SOF_BIT           equ 0020h      ; Start of Frame (not used)
; EOF_BIT           equ 0040h      ; End of Frame
; OVERRUN_ERR      equ 0080h      ; Frame Overrun bit
; STATUS_ERROR     equ 8000h      ; if reset, buffer may be used
; FRAMING_ERR OR CRC_ERR OR ABORT OR ALIGN_ERR OR
; DES_ERR OR OVERRUN_ERR
```

```
DebugMessage macro mask, message
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push offset message
    push DPCTScreen
    call OutputToScreen
    lea esp, [esp + (2 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage1 macro mask, message, parm0
```

```
    local DebugMessageExit
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm0
    push offset message
    push DPCTScreen
    call OutputToScreen
    lea esp, [esp + (3 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage2 macro mask, message, parm0, parm1
```

```
    local DebugMessageExit
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm1
    push parm0
    push offset message
    push DPCTScreen
    call OutputToScreen
    lea esp, [esp + (4 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage3 macro mask, message, parm0, parm1, parm2
```

```
    local DebugMessageExit
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm2
    push parm1
    push parm0
    push offset message
    push DPCTScreen
    call OutputToScreen
    lea esp, [esp + (5 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage4 macro mask, message, parm0, parm1, parm2, parm3
```

```
    local DebugMessageExit
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm3
    push parm2
    push parm1
    push parm0
    push offset message
    push DPCTScreen
    call OutputToScreen
    lea esp, [esp + (6 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
    endm
```

```
DebugMessage5 macro mask, message, parm0, parm1, parm2, parm3, parm4
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push parm4
    push parm3
    push parm2
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (7 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
```

```
    endm
```

```
DebugMessage6 macro mask, message, parm0, parm1, parm2, parm3, parm4, parm5
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    xor eax, eax
    mov al, parm5
    push eax
```

```
    mov al, parm4
```

```
    push eax
```

```
    mov al, parm3
```

```
    push eax
```

```
    mov al, parm2
```

```
    push eax
```

```
    mov al, parm1
```

```
    push eax
```

```
    mov al, parm0
```

```
    push eax
```

```
    push offset message
```

```
    push DPCScreen
```

```
    call OutputToScreen
```

```
    lea esp, [esp + (8 * 4)]
```

```
    pop edx
```

```
    pop ecx
```

```
    pop eax
```

```
DebugMessageExit:
```

```
    endm
```

```
SLOW
```

```
macro
```

```
    eax
    push
    in al, 61h
    in al, 61h
    in al, 61h
    pop
    eax
    endm
```

```
BufferStruct struc
```

```
    BufPtr dd 0 ; Pointer to buffer
    DataSize dd 0 ; Size of buffer
BufferStruct ends
```

```
LAST_EOF equ 80000000h ; or with DataSize
```

```
MAX_APPL_RBD equ 350
```

```
MAX_CHAN equ 10
```

```
RBD_NOT_USED equ -1
```

```
MAX_CONF_ADDR equ 8
```

```
MAX_ADDR equ 16
```

```
RX_CNTL struc
```

```
    RxChannel dd 0 ; Channel ID
```

```
    RxESR dd 0 ; ESR to call when a packet
```

```
    ends ; is received.
```

```
RX_CNTL ends
```

```
RX_FLAG_STATS_ONLY equ 1
```

```
ChannelConfig struc
```

```
    CfgChannel dd 0 ; Channel ID
```

```
    CfgESR dd 0 ; max # of buffers used
```

```
    CfgNumAddresses dd 0 ; Number of filter addrs
```

```
    CfgAddress db 6 dup (0) ; that is to follow
```

```
    CfgGroupKey db 8 dup (0) ; list of addrs
```

```
    CfgElementKey db 8 dup (0) ; Group keys
```

```
    ends ; Element keys
```

```
ChannelConfig ends
```

```
FilterStruct struc
```

```
    FilterAddress db 6 dup (0) ; 48 bit address
```

```
    FilterChannel dd 2 dup (0) ; Align next field
```

```
    FilterCmdBlkIndex dd 0 ; Channel ID
```

```
    FilterTotalCount dd 0 ; RISC SW cmd blk index
```

```
    FilterSeqCount dd 0 ; # of frames rxed
```

```
    FilterSeqNum dd 0 ; # of out-of-seq frames
```

```
    ends ; next seq expected
```

```
FilterStruct ends
```

```
EblkStruct struc
```

```
    EblkCmd dw 0 ;
```

```
    EblkPortID dw 0 ;
```

```
    EblkNotUsed dw 0 ;
```

```
    EblkAddress db 6 dup (0) ;
```

```
    EblkGroupKey db 8 dup (0) ;
```

```
    EblkElemKey db 8 dup (0) ;
```

```
    ends ;
```

```
EblkStruct ends
```



```

EblkBusyFlags      db      32 dup (0)

BufferCount        dd      0
WatchBufferCount   dd      0
WatchOldRejected   dd      0
; ; LocalMipStats   db      0
AgentSendRoutine   dd      0
AgentRemoveRoutine dd      0

GotInterrupt       dd      0
IOEnableValue      dd      0
; *****
; Error Counters.
; *****
; *****
; *****
StatisticsVersion  db      03, 00
GenericVariableCount dw      (RxAbortFrameAlignment + 4 - TotalTxPacketCount)
/ 4
NotSupportedMask0  dd      11110111100100000000000000000011b

TotalTxPacketCount dd      0 ; 1 - (Used by MSM)
TotalRxPacketCount dd      0 ; 1 - (Used by MSM)
NoECBAVAILABLECount dd      0 ; 1 - (Used by MSM)
PacketTxTooBigCount dd      0 ; 1 - not used
PacketTxTooSmallCount dd      0 ; 1 - not used
PacketRxOverflowCount dd      0 ; 0 - Used by driver
PacketRxTooBigCount dd      0 ; 1 - not used
PacketRxTooSmallCount dd      0 ; 1 - not used
PacketTxMiscErrorCount dd      0 ; 1 - not used
PacketRxMiscErrorCount dd      0 ; 1 - not used
RetryTxCount       dd      0 ; 0 - Used by driver
ChecksumErrorCount dd      0 ; 0 - Used by driver
HardwareRxMismatchCount dd      0 ; 1 - (Used by MSM)
TotalTxORByteCountLow dd      0 ; 0 - Used by MSM
TotalTxORByteCountHigh dd      0 ; 0 - Used by MSM
TotalRxORByteCountLow dd      0 ; 0 - Used by MSM
TotalRxORByteCountHigh dd      0 ; 0 - Used by MSM
TotalGroupAddrTxCount dd      0 ; 0 - Used by MSM
TotalGroupAddrRxCount dd      0 ; 0 - Used by MSM
AdapterResetCount  dd      0 ; 0 - Used by driver
AdapterOpTimeStamp dd      0 ; 0 - Used by MSM
QDepth            dd      0 ; 0 - Used by MSM

TxOKSingleCollision dd      0 ; 0 - Used by driver
TxOKMultipleCollisions dd      0 ; 0 - Used by driver
TxOKButDeferred      dd      0 ; 0 - not used
TxAbortLateCollision dd      0 ; 0 - Used by driver
TxAbortExcessCollisions dd      0 ; 0 - Used by driver
TxAbortCarrierSense  dd      0 ; 0 - Used by driver
TxAbortExDeferral    dd      0 ; 0 - not used
RxAbortFrameAlignment dd      0 ; 0 - Used by driver

CustomVariableCount dw      (MipsFullBufsRejec + 4 - SignalQuality)
/ 4

SignalQuality      dd      0 ; Signal Quality
from adapter
RxFreq            dd      0 ; Recieve Freque
ncy
LargestRx         dd      0
NumberLargeRx     dd      0

AveLargeRx        dd      0
MipsRxEnables     dd      0 ; number of time
s the RSW has gotten a rx
; enable cmd.
MipsRxdisables    dd      0 ; number of rx disable c
ommands.
MipsFramesAccepted dd      0 ; total number of frames accepted.
MipsNoFilterMatch  dd      0 ; number of frames rejected due
to no filter match.
MipsZeroAddrFrames dd      0 ; number of frames rejected due to an
; address of all zeroes or RBD shortage
MipsDisablesRejed  dd      0 ; number of frames which had to
be rejected while zeroed.
MipsFullBufsRejec dd      0 ; number of frames rejected beca
use the
; adapter was close to running out of buffers (RBDS
).

VectorToTheStrings dd      offset DiagnosticsStrings
TotalLargeRx       dd      0
AlignEndVA         db      ((4 - offset (AlignEndVA and 3)) and 3) + 4 dup
(?) ; Align 4 for MOVSD
DriverAdapterDataSpace ends
IORbBase           equ      IOStatus
IOTuningLowAddr     equ      IOCountNomLowAddr
IOTuningHighAddr    equ      IOCountNomHighAddr
IORelSgfAddr        equ      IOCountDeltaAddr
IOMaxSgfAddr        equ      IOSweepRateAddr
IOStatusAddr        equ      IODAdOffsetControlAddr

subttl -- OSDATA Data Segment --
page
assume cs: OSCODE, ds: OSDATA, es: OSDATA, ss: OSDATA
; *****
; The following variables are common to the entire driver.
; *****
; *****
; OSDATA segment rw public 'DATA'
HSMSSPEC           db      'HSM_SPEC_VERSION: 3.2', 0
; *****
; *****
; Statistic Diagnostic Strings.
; *****
; *****
; *****
DiagnosticsStrings dw      (EndOfStrings-DiagnosticsStrings)
db      'Signal Quality', 0
db      'RX Frequency', 0
db      'Largest Received IP Frame', 0
db      'Number of Receive IP Frames over 400', 0
db      'Ave size of Receive IP Frames over 400', 0
db      'Mips Rx enable commands', 0

```

```

db      'Mips Rx disable commands', 0
db      'Mips frames accepted', 0
db      'Mips frames rejected from no filter match', 0
db      'Mips frames rejected from zero address', 0
db      'Mips frames rejected from adapter disabled', 0
db      'Mips frames rejected from low buffer pool', 0

db      0,0

EndOfStrings equ $
;*****
; Driver Parameter Block to pass to MSM.
;*****
;
; align 4
DriverParameterBlock label dword
DriverParameterSize dd DriverParameterBlockSize
DriverStackPointer dd 0
DriverModuleHandle dd 0
DriverBoardPointer dd 0
DriverAdapterPointer dd 0
DriverConfigTemplatePtr dd DriverConfigTemplate
DriverFirmwareSize dd 0
DriverFirmwareBuffer dd 0
DriverNumKeywords dd 2
DriverKeywordText dd DPCKeywordText
DriverKeywordTextLen dd DPCTextLen
DriverProcessKeywordTab dd DPCProcessKeywordTab
DriverAdapterDataSpaceSize dd SIZE DriverAdapterDataSpace
DriverAdapterTemplate dd DriverAdapterDataSpaceTemplate
DriverStatisticsTable dd StatisticsVersion
DriverEndOfChainFlag dd 0
DriverMaxMulticast dd 0
DriverNeedsBelow16Meg dd 0
DriverAESPtr dd 0
DriverCallBackPtr dd 0
DriverISRPtr dd 0
DriverMulticastChangePtr dd 0
DriverPollPtr dd 0
DriverResetPtr dd 0
DriverSendPtr dd 0
DriverShutdownPtr dd 0
DriverTxTimeoutPtr dd 0
DriverPromiscuousChangePtr dd 0
DriverStatisticsChangePtr dd 0
DriverRxLookAheadChangePtr dd 0
DriverManagementPtr dd 0
DriverEnableInterruptPtr dd 0
DriverDisableInterruptPtr dd 0
DriverParameterBlockSize equ $ - DriverParameterBlock
;*****
; Driver Management Dispatch Jump Table.
;*****
;
; ManagementJumpTable label dword
; dd offset GetMipsStats
; dd offset OpenChannel
; dd offset CloseChannel

```

```

dd      offset GetSN
dd      offset SignText
dd      offset AddAddress
dd      offset DeleteAddress
dd      offset RegisterAgentSendRoutine
dd      offset BadParametersExit
dd      offset RegisterAgent
dd      offset ReturnTCBCompleteRoutine
LastManagementFunction equ (($ - ManagementJumpTable) / 4) - 1
;*****
; Copy of Virtual Adapter Data area to be copied at
; initialization.
;*****
;
; DriverAdapterDataSpaceTemplate DriverAdapterDataSpace <>
DriverConfigTemplate db 'HardwareDriverMLID
Signature          01          ; [ebx].MLIDCFG_MajorVersion
                  12          ; [ebx].MLIDCFG_MinorVersion
                  6 dup (0FFh) ; [ebx].MLIDNodeAddress
                  dw 0010010001001001b ; [ebx].MLIDNodeFlags
                  dw 0000      ; [ebx].MLIDBoardNumber
                  dd 00000000 ; [ebx].MLIDBoardInstance
                  dd 00000000 ; [ebx].MLIDMaximumSize
                  dd 00000000 ; [ebx].MLIDMaxRecvSize
                  dd 00000000 ; [ebx].MLIDRecvSize
                  dd 00000000 ; [ebx].MLIDCardName
                  dd DriverNICShortName ; [ebx].MLIDShortName
                  dd 00000000 ; [ebx].MLIDFrameType
                  dw 0000      ; [ebx].MLIDReserved0
                  dw 0000      ; [ebx].MLIDFrameID
                  dd 0001      ; [ebx].MLIDTransportTime
                  dd 00000000 ; [ebx].MLIDRouteHandler
                  dw 10        ; [ebx].MLIDLineSpeed
                  dw 0000      ; [ebx].MLIDLookAheadSize
                  db 8 dup (00h) ; [ebx].MLIDReserved
                  MLID_MAJOR_VERSION ; [ebx].MLIDMajor
                  MLID_MINOR_VERSION ; [ebx].MLIDMinor
                  0010b        ; [ebx].MLIDFlags
                  dw 0010      ; [ebx].MLIDSendRetries
                  dd 00000000 ; [ebx].MLIDLink
                  dw 0000      ; [ebx].MLIDSharingFlags
                  dw 0000      ; [ebx].MLIDSlot
                  dw 02c0h, 40h, 0, 0 ; [ebx].MLIDPortsAndLengths
                  dd 00000000 ; [ebx].MLIDMemoryDecode0
                  dw 0000      ; [ebx].MLIDLength0
                  dd 00000000 ; [ebx].MLIDMemoryDecode1
                  dw 0000      ; [ebx].MLIDLength1
                  db 03, 0FFh ; [ebx].MLIDInterrupt
                  db 0FFh, 0FFh ; [ebx].MLIDMAUsage
                  dd 00000000 ; [ebx].MLIDResourceTag
                  dd 00000000 ; [ebx].MLIDConfiguration
                  dd 00000000 ; [ebx].MLIDCommandString
                  db 18 dup (0) ; [ebx].MLIDLogicalName
                  dd 00000000 ; [ebx].MLIDLinearMemory0
                  dd 00000000 ; [ebx].MLIDLinearMemory1
                  dw 0000      ; [ebx].MLIDChannelNumber
                  dw 6 dup (0) ; [ebx].MLIDIOReserved
                  Message      DriverNICShortName, 'DPC'

```


EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module.
It can be called at process or interrupt time.See Also: EThERTSM\EtherTSMAddMulticastAddress
EThERTSM\EtherTSMDeleteMulticastAddress
EThERTSM\EtherTSMUpdateMulticast

END_MANUAL_ENTRY

DriverManagement proc

First reset Multicast Address Registers.

cmp dword ptr [esi].RProtocolID+0, 'ICRD' ; ID=0 == 'DRCT'?
jne BadParametersExit ; Jump if not
cmp word ptr [esi].RProtocolID+4, 'CP' ; ID=4 == 'PC'?
jne BadParametersExit ; Jump if notmovzx ecx, [esi].RLogicalID ; ECX = Function
cmp ecx, LastManagementFunction ; Supported?
ja BadParametersExit

jmp ManagementJumpTable[ecx * 4]

BadParametersExit:

mov eax, BadParameters
ret

DriverManagement endp

subttl -- SignText --
page

BEGIN_MANUAL_ENTRY(SignText, DPC/API/SIGNTEXT)

Name: SignText

Description: This routine is called by DriverManagement to
sign the text passed in.On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX DestroyedEBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

See Also:

END_MANUAL_ENTRY

SignText public SignText
proc

mov esi, [esi].RPacketOffset

mov edi, [esi+0]

or edi, edi

je SignTextError

cmp dword ptr [esi+8], 0

je SignTextError

cli

mov edx, [ebp].IOMsgRamPtr

mov eax, 18800h / 2

out dx, ax

mov ecx, [esi+4]

shr ecx, 1

mov edx, [ebp].IOMsgRam

inc ecx

SendStringLoop:

mov ax, [edi]

out dx, ax

add edi, 2

dec ecx

jne SendStringLoop

sti

cli

mov edx, [ebp].IOMsgRamPtr

mov eax, (18100h + (14 * size EblkStruct) + 4) / 2

out dx, ax

mov edx, [ebp].IOMsgRam

mov eax, [esi + 4]

out dx, ax

mov edx, [ebp].IOMsgRamPtr

mov eax, (18100h + (14 * size EblkStruct)) / 2

out dx, ax

mov edx, [ebp].IOMsgRam

mov eax, 10h

out dx, ax

sti

mov ecx, 10

mov edi, 10h

```
WaitForCommandDone:
```

```
    push    ecx
    mov     eax, [ebp].TimerTag
    push    eax
    push    2
    call    DelayMyself
    add     esp, (2 * 4)
    pop     ecx
```

```
    cli
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, (18100h + (14 * size Eb1kStruct)) / 2
    out     dx, ax
```

```
    mov     edx, [ebp].IOMsgRam
    in      ax, dx
    sti
    cmp     ax, 0eeh
    je      SignTextError
```

```
    cmp     ax, 1lh
    je      ReadyToGetSignature
```

```
    dec     ecx
    jne     WaitForCommandDone
```

```
ReadyToGetSignature:
```

```
    mov     edi, [esi+8]
    cli
```

```
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18790h / 2
    out     dx, ax
```

```
    mov     ecx, 4
    mov     edx, [ebp].IOMsgRam
```

```
GetSignatureLoop:
```

```
    in      ax, dx
    mov     [edi], ax
    add     edi, 2
    dec     ecx
    jne     GetSignatureLoop
```

```
    sti
    mov     dword ptr [esi+4], 8
    xor     eax, eax
    ret
```

```
SignTextError:
```

```
    mov     eax, -1
    ret
```

```
SignText
```

```
    endp
    subttl  -- GetSN --
    page
```

```
*****\
```

```
; BEGIN_MANUAL_ENTRY( GetSN, DPC/API/GETSN )
```

```
; Name: GetSN
```

```
; Description: This routine is called by DriverManagement to
;              return the adapters serial number.
```

```
; On Entry:
```

```
    EAX    N/A
    EBX    Frame Data Space
    ECX    N/A
    EDX    N/A
    EBP    Adapter Data Space
    ESI    ECB
    EDI    N/A
```

```
    Note: Interrupts are in any state.
```

```
; On Return:
```

```
    EAX    Destroyed
    EBX    Preserved
    ECX    Destroyed
    EDX    Destroyed
    EBP    Preserved
    ESI    Preserved
    EDI    Preserved
```

```
    Flags:
```

```
    Note: Interrupts preserved.
```

```
    Remarks: This routine is called by DriverManagement.
             It is called at process time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
*****\
```

```
    public GetSN
    GetSN proc
```

```
    mov     esi, [esi].RPacketOffset
```

```
    cli
```

```
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18760h / 2
    out     dx, ax
```

```
    mov     ecx, 3
    mov     edx, [ebp].IOMsgRam
```

```
GetSNLoop:
```

```
    in      ax, dx
    mov     [esi], ax
    add     esi, 2
    dec     ecx
    jne     GetSNLoop
```

```
    sti
    mov     [esi], cl
    ret
```

```
GetSN    endp
    subttl  -- CloseChannel --
    page
```

```
*****\
```

```
; BEGIN_MANUAL_ENTRY( CloseChannel, DPC/API/CLSCHAN )
```

```
; Name: CloseChannel
```

```
; Description: This routine is called by DriverManagement to
```

```

; close the specified channel.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  ECB
;            EDI  N/A
;
; Note:  Interrupts are in any state.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:  Interrupts preserved.
;
; Remarks:  This routine is called by DriverManagement.
;           It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
; *****
;
; public CloseChannel
; CloseChannel proc
;
;     mov     esi, [esi].RPacketOffset    ; ESI -> config structure
;     mov     esi, [esi]                  ; ESI = channel
;
;     cmp     esi, MAX_CHAN
;     ja      CloseChannelError
;     lea     edi, [ebp].RxControl[esi*8]
;     cmp     [edi].RxChannel, RBD_NOT_USED
;     je      CloseChannelError
;
;     mov     [edi].RxChannel, RBD_NOT_USED
;     mov     [edi].RxESR, 0
;
;     mov     ecx, MAX_ADDR
;     lea     edi, [ebp].Filter
;
; CloseChannelCloseFilterLoop:
;     cmp     [edi].FilterChannel, esi
;     jne     CloseChannelCloseFilterNext
;
;     [edi].FilterChannel, RBD_NOT_USED
;     mov     eax, [edi].FilterCmdblkIndex
;     mov     [ebp].EblkBusyFlags(eax), 0
;
;     push    ecx
;     mov     ecx, size EblkStruct
;     mul     ecx
;     mov     edx, [ebp].IOMsgRamPtr
;     add     eax, 18100h
;     shr     eax, 1
;     push    eax
;     push    eax, 3
;     add

```

```

;
; out     dx, ax
;
; mov     edx, [ebp].IOMsgRam
; xor     eax, eax
; out     dx, ax
; out     dx, ax
; out     dx, ax
;
; pop     eax
; pop     ecx
; mov     edx, [ebp].IOMsgRamPtr
; out     dx, ax
; mov     edx, [ebp].IOMsgRam
; mov     eax, 1
; out     dx, ax
;
; CloseChannelCloseFilterNext:
;     add     edi, size FilterStruct
;     dec     ecx
;     jne     CloseChannelCloseFilterLoop
;
;     xor     eax, eax
;     ret
;
; CloseChannelError:
;     mov     eax, -1
;     ret
;
; CloseChannel     endp
;     subttl  -- AddAddress --
;     page
; *****
;
; BEGIN_MANUAL_ENTRY( AddAddress, DPC/API/ADDADRS )
;
; Name:  AddAddress
;
; Description:  This routine is called by DriverManagement to
;              add the address passed in.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  ECB
;            EDI  N/A
;
; Note:  Interrupts are in any state.
;
; On Return:  EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:  Interrupts preserved.
;
; Remarks:  This routine is called by DriverManagement.
;           It is called at process time.
;
;
; *****

```

```
; See Also:
;
; END_MANUAL_ENTRY
;
;*****
;
; public Address
; Address
;
; mov esi, [esi].RPacketOffset
; mov eax, [esi].CfgChannel
; cmp ecx, MAX_CHAN
; ja OpenChannelError
;
; lea edi, [ebp].RxControl[eax*8]
; cmp [edi].RxChannel, RBD_NOT_USED
; je OpenChannelError
;
; ; Fall thru to AddAddr
;
;
; Address      endp
; subttl  -- AddAddr --
; page
;
;*****\
;
; BEGIN_MANUAL_ENTRY( AddAddr, DPC/API/ADDADDR )
;
; Name:      AddAddr
;
; Description: This routine is called by AddAddress and OpenChannel
;              to add the address to the adapter.
;
; On Entry:  EAX      N/A
;            EBX      Frame Data Space
;            ECX      N/A
;            EDI      N/A
;            EBP      Adapter Data Space
;            ESI      ECB
;            EDI      N/A
;
; Note:      Interrupts are in any state.
;
; On Return: EAX      Destroyed
;            EBX      Preserved
;            ECX      Destroyed
;            EDX      Destroyed
;            EBP      Preserved
;            ESI      Preserved
;            EDI      Preserved
;
; Flags:
;
; Note:      Interrupts preserved.
;
; Remarks:   This routine is called by AddAddress and OpenChannel.
;            It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****/
```

```
public AddAddr
AddAddr proc

    mov     eax, dword ptr [esi].CfgAddress
    DebugMessage1 DEBUG_IOCTL, AddAddrMsg, eax

    mov     ecx, [esi].CfgNumAddresses
    ; ECX = Number 0

f Addrs
AddAddrLoop:
; ; First make sure this address is not a duplicate
;
    lea     edi, [ebp].Filter
    mov     edx, MAX_ADDR
AddAddrDuplicateLoop:
    cmp     [edi].FilterChannel, RBD_NOT_USED
    je      AddAddrDuplicateNext
    mov     eax, dword ptr [edi].FilterAddress
    cmp     eax, dword ptr [esi].CfgAddress
    jne     AddAddrDuplicateNext
    mov     ax, word ptr [edi].FilterAddress+4
    cmp     ax, word ptr [esi].CfgAddress+4
    jne     AddAddrDuplicateNext
    mov     eax, -1
    ret
AddAddrDuplicateNext:
    add     edi, size FilterStruct
    dec     edx
    jne     AddAddrDuplicateLoop
; ; Find an empty slot in the filter table
;
    lea     edi, [ebp].Filter
    xor     edx, edx
AddAddrFindEmptyLoop:
    cmp     [edi].FilterChannel, RBD_NOT_USED
    je      AddAddrFindEmptyFound
    add     edi, size FilterStruct
    inc     edx
    cmp     edx, MAX_ADDR
    jb      AddAddrFindEmptyLoop
    mov     eax, -1
    ret
AddAddrFindEmptyFound:
    mov     eax, [esi].CfgChannel
    mov     [edi].FilterChannel, eax
    mov     eax, dword ptr [esi].CfgAddress
    mov     dword ptr [edi].FilterAddress, eax
    mov     ax, word ptr [esi].CfgAddress+4
    mov     word ptr [edi].FilterAddress+4, ax
    mov     [edi].FilterTotalCount, 0
    mov     [edi].FilterSeqCount, 0
    mov     [edi].FilterSeqNum, 0
    mov     edx, 16
    mov     eax, 31
    test    [esi].CfgAddress+0, 02h
    jnz     AddAddrFindEblkLoop
    mov     edx, 2
    mov     eax, 13
;
;*****/
```

```

AddAddrFindEblkLoop:
    cmp     [ebp].EblkBusyFlags[edx], 0
    je      AddAddrFindEblkFound
    inc     ecx
    cmp     edx, eax
    jbe     AddAddrFindEblkLoop
    mov     eax, -1
    ret

AddAddrFindEblkFound:
    [ebp].EblkBusyFlags[edx], 1
    [edi].FilterCmdBkIndex, edx

    mov     eax, [edi].FilterChannel
    lea     edi, [ebp].Eblk
    mov     [edi].EblkCmd, 0
    mov     [edi].EblkPortID, ax
    mov     ax, word ptr [esi].CfgAddress
    xchg    ah, al
    mov     word ptr [edi].EblkAddress, ax
    mov     ax, word ptr [esi].CfgAddress+2
    xchg    ah, al
    mov     word ptr [edi].EblkAddress+2, ax

    cmp     edx, 16
    jae     AddAddrIsBypass
    mov     eax, dword ptr [esi].CfgGroupKey
    mov     dword ptr [edi].EblkGroupKey, eax
    mov     eax, dword ptr [esi].CfgGroupKey+4
    mov     dword ptr [edi].EblkGroupKey+4, eax
    mov     eax, dword ptr [esi].CfgElementKey
    mov     dword ptr [edi].EblkElementKey, eax
    mov     eax, dword ptr [esi].CfgElementKey+4
    mov     dword ptr [edi].EblkElementKey+4, eax

AddAddrIsBypass:
    pushfd
    cli
    push    ecx
    mov     eax, edx
    mov     ecx, size EblkStruct
    mul     ecx
    mov     edx, [ebp].IOMsgRamPtr
    add     eax, 18100h
    shr     eax, 1
    push    eax
    push    esi
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     ecx, size EblkStruct / 2
    mov     esi, edi
    cld
    lodsw
    out     dx, ax
    dec     ecx
    jne     AddAddrCopyEblk
    pop     esi
    pop     eax
    pop     ecx
    mov     edx, [ebp].IOMsgRamPtr
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     eax, 1

```

```

    out     dx, ax
    popfd
    dec     ecx
    jne     AddAddrLoop
    xor     eax, eax
    ret

AddAddr endp
    subttl  -- DeleteAddress --
page
; *****
; BEGIN_MANUAL_ENTRY( DeleteAddress, DPC/API/DELADDR )
; Name:      DeleteAddress
; Description: This routine is called by DriverManagement to
;              delete the address passed in.
; On Entry:   EAX N/A
;              EBX Frame Data Space
;              ECX N/A
;              EDX N/A
;              EBP Adapter Data Space
;              ESI ECB
;              EDI N/A
; Note:       Interrupts are in any state.
; On Return:  EAX Destroyed
;              EBX Preserved
;              ECX Destroyed
;              EDX Destroyed
;              EBP Preserved
;              ESI Preserved
;              EDI Preserved
; Flags:
; Note:       Interrupts preserved.
; Remarks:    This routine is called by DriverManagement.
;              It is called at process time.
; See Also:
; END_MANUAL_ENTRY
; *****
; public DeleteAddress
DeleteAddress proc
    mov     esi, [esi].RPacketOffset
    mov     eax, [esi].CfgChannel
    cmp     eax, MAX_CHAN
    ja      DeleteAddressError
    lea     edi, [ebp].RxControl[eax*8]
    cmp     [edi].RxChannel, RBD_NOT_USED
    je      DeleteAddressError
    mov     ecx, [esi].CfgNumAddresses
    ; ESI -> config structure
    ; over the max?
    ; jump if it is.

```

```

cmp ecx, MAX_CONF_ADDR
ja DeleteAddressError

```

```

lea ebx, [esi].CfgAddress
DeleteAddressLoop:

```

```

mov ch, MAX_ADDR
lea edi, [ebp].Filter
DeleteAddressFilterLoop:
mov eax, dword ptr [ebx+0]
cmp eax, dword ptr [edi].FilterAddress+0
jne DeleteAddressNextFilter
mov ax, word ptr [ebx+4]
cmp ax, word ptr [edi].FilterAddress+4
jne DeleteAddressNextFilter
mov eax, [esi].CfgChannel
cmp eax, [edi].FilterChannel
jne DeleteAddressNextFilter

```

```

[edi].FilterChannel, RBD_NOT_USED
eax, [edi].FilterCmdblkIndex
[ebp].EblkBusyFlags[eax], 0

```

```

pushfd
cli
push ecx
mov ecx, size EblkStruct
mul ecx
mov edx, [ebp].IOMsgRamPtr
add eax, 18100h
shr eax, 1
add eax, 3
push eax
out dx, ax
mov edx, [ebp].IOMsgRam
xor eax, eax
out dx, ax
out dx, ax
out dx, ax

```

```

pop eax
pop ecx
sub eax, 3
mov edx, [ebp].IOMsgRamPtr
out dx, ax
mov edx, [ebp].IOMsgRam
mov eax, 1
out dx, ax
popfd

```

```

DeleteAddressNextFilter:
add edi, size FilterStruct
dec ch
jne DeleteAddressFilterLoop

```

```

DeleteAddressNext:
add ebx, 6
dec cl
jne DeleteAddressLoop
xor eax, eax
ret

```

```

DeleteAddressError:
mov eax, -1

```

```

ret
DeleteAddress endp
subttl -- RegisterAgentSendRoutine --
page

```

```

; *****
; BEGIN_MANUAL_ENTRY( RegisterAgentSendRoutine, DPC/API/DELADDR )
;
; Name: RegisterAgentSendRoutine
;
; Description: This routine is called by DriverManagement to
; register a slip send routine. The first dword in the first
; ECB fragment points to the Slip Send routine to use. To
; deregister the send routine, set the dword to a NULL.

```

```

; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A

```

```

; Note: Interrupts are in any state.

```

```

; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved

```

```

; Flags:

```

```

; Note: Interrupts preserved.

```

```

; Remarks: This routine is called by DriverManagement.
; It is called at process time.

```

```

; See Also:

```

```

; END_MANUAL_ENTRY

```

```

; *****
; RegisterAgentSendRoutine proc

```

```

RegisterAgentSendRoutine proc

```

```

e mov esi, [esi].RPacketOffset ; esi -> config structur
mov eax, [esi]
; EAX -> Slip Send Routine Address
mov [ebp].AgentSendRoutine, eax ; Save it for later
xor eax, eax
ret

```

```

RegisterAgentSendRoutine endp
subttl -- RegisterAgent --
page

```

```

; *****

```

```

; BEGIN_MANUAL_ENTRY( RegisterAgent, DPC/API/REGAG )

```

; Name: RegisterAgent

; Description: This routine is called by DriverManagement to register package delivery/Internet agent. The first dword in the first ECB fragment points to the Remove routine that we must call before we are removed.

; On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

; Note: Interrupts are in any state.

; On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

; Flags:

; Note: Interrupts preserved.

; Remarks: This routine is called by DriverManagement. It is called at process time.

; See Also:

; END_MANUAL_ENTRY

; *****/

; RegisterAgent proc

```

e      mov     esi, [esi].RPacketOffset          ; ESI -> config structur
      mov     eax, [esi]
      mov     [ebp].AgentRemoveRoutine, eax    ; Save it for later
      xor     eax, eax
      ret

```

```

RegisterAgent endp
      subttl  -- ReturnTCBCompleteRoutine --
      page

```

; *****/

; BEGIN_MANUAL_ENTRY(ReturnTCBCompleteRoutine, DPC/API/RETADS)

; Name: ReturnTCBCompleteRoutine

; Description: This routine is called by DriverManagement to return the TCB Complete routine. The first dword in the first ECB fragment points to a LONG which we will return with a pointer to the TCB complete routine.

; On Entry: EAX N/A
EBX Frame Data Space
ECX N/A

; EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

; Note: Interrupts are in any state.

; On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

; Flags:

; Note: Interrupts preserved.

; Remarks: This routine is called by DriverManagement. It is called at process time.

; See Also:

; END_MANUAL_ENTRY

; *****/

; if TIMESTAMP

```

extrn  GetHighResolutionTimer: near
extrn  HighResolutionTimer: dword
public DPCtimestamp
DPCtimestamp
proc

```

```

Cpush
call

```

; GetHighResolutionTimer

```

and     eax, 00ffffffh
mov     edx, timestamp_index
mov     ecx, [esp + Parm0]
shl     ecx, 24
or      eax, ecx
mov     [edx], eax

```

```

add     edx, 4
cmp     edx, timestamp_end
jae     short DPCtimestampBegin
DPCtimestampExit:
mov     timestamp_index, edx
mov     dword ptr [edx], '$$$'
Cpop
ret

```

```

DPCtimestampBegin:
mov     edx, timestamp_begin
jmp     DPCtimestampExit

```

```

DPCtimestamp endp
endif

```

```

subttl  -- DriverTCBComplete --
page

```

; DriverTCBComplete proc

; Cpush


```

mov     esi, (esp + Parm0)
mov     ebp, OurAdapterDataSpace
inc     [ebp].MSMTxFreeCount
call    EthernetsFastSendComplete

test    [ebp].MSMStatusFlags, SHUTDOWN
jne     GetNextSendExit

GetNextSendLoop:
    test    [ebp].MSMStatusFlags, TXQUEUED
    jnz     short GetNextSendGetIt

GetNextSendExit:
    CPop
    ret

```

```

GetNextSendGetIt:
    call    EthernetsGetNextSend
    jne     short GetNextSendExit
    call    DriverSend
    jmp     GetNextSendLoop
; See if we have any more

```

```

DriverTCBComplete    endp

ReturnTCBCompleteRoutine proc
    mov     esi, [esi].RPacketOffset
    mov     dword ptr [esi], offset DriverTCBComplete
    xor     eax, eax
    ret

```

```

ReturnTCBCompleteRoutine endp
subttl    -- OpenChannel --
page
; *****\
; BEGIN_MANUAL_ENTRY( OpenChannel, DPC/API/OPENCHAN )
; Name:      OpenChannel
; Description: This routine is called by DriverManagement to
;              open a channel on the adapter to receive packets
;              from.
; On Entry:   EAX    N/A
;             EBX    Frame Data Space
;             ECX    N/A
;             EDX    N/A
;             EBP    Adapter Data Space
;             ESI    ECB
;             EDI    N/A
; Note:      Interrupts are in any state.
; On Return:  EAX    Destroyed
;             EBX    Preserved
;             ECX    Destroyed
;             EDX    Destroyed
;             EBP    Preserved
;             ESI    Preserved
;             EDI    Preserved
; Flags:

```

```

; Note:      Interrupts preserved.
; Remarks:   This routine is called by DriverManagement.
;             It is called at process time.
; See Also:
; END_MANUAL_ENTRY
; *****\
; public    OpenChannel
; proc

```

```

    mov     esi, [esi].RPacketOffset
; Don't open if we don't have signal lock.
    cmp     [ebp].SignalQuality, 200
    jbe     OpenChannelError
; Find an empty RxControl entry
    xor     ecx, ecx
    lea     edi, [ebp].RxControl
; ECX = index
; EDI -> Rx Control Structures
FindEmptyRBDLoop:
    cmp     [edi].RxChannel, RBD_NOT_USED
    jne     FindEmptyRBDNext
; Found one. Initialize it before adding addresses.
    mov     [edi].RxChannel, ecx
    mov     [esi].CfgChannel, ecx
    mov     eax, [esi].CfgESR
    mov     [edi].RxESR, eax
    push    edi
    call    AddAddr
    pop     edi
    or      eax, eax
    jne     OpenChannelDidntAdd
    ret

```

```

OpenChannelDidntAdd:
    mov     [edi].RxChannel, RBD_NOT_USED
    mov     [edi].RxESR, 0
    mov     eax, -1
    ret

```

```

FindEmptyRBDNext:
    inc     ecx
    add     edi, size RX_CNTL
    cmp     ecx, MAX_CHAN
    jbe     FindEmptyRBDLoop

```

```

OpenChannelError:
    mov     eax, -1
    ret

```

```

OpenChannel    endp
subttl    -- RefreshMipsStats --
page
; *****\
; BEGIN_MANUAL_ENTRY( RefreshMipsStats, DPC/API/REFSHMIPS )

```


bytes 0-5 = Multicast Address.
bytes 6-7 = Entry used(Non zero if used).

On Entry: EAX N/A
EBX N/A
ECX # of Entries in Table(0 if empty)
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also: EHERTSM\EtherTSMAddMulticastAddress
ETHERTSM\EtherTSMDeleteMulticastAddress
ETHERTSM\EtherTSMUpdateMulticast

END_MANUAL_ENTRY

DriverMulticastChange proc

First reset Multicast Address Registers.

ret

DriverMulticastChange endp

subttl -- DriverPromiscuousChange --
page

BEGIN_MANUAL_ENTRY(DriverPromiscuousChange, DPC/API/PROMISCU)

Name: DriverPromiscuousChange

Description: This routine will enable/disable the Promiscuous Mode.

On Entry: EAX N/A
EBX N/A
ECX 0 to disable the Promiscuous mode
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also: EHERTSM\EtherTSMPromiscuousChange
END_MANUAL_ENTRY

DriverPromiscuousChange proc

ret

DriverPromiscuousChange endp
subttl -- CalculatedDriftDelta --
page

BEGIN_MANUAL_ENTRY(CalculatedDriftDelta, DPC/API/CALCDD)

Name: CalculatedDriftDelta

Description: Acquisition State Routine.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI N/A
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by InitState.
It can be called at process or interrupt time.

See Also:

; END_MANUAL_ENTRY

; *****
; public CalculateDriftDelta
; CalculateDriftDelta procmov edi, [ebp].Drift
cmp edi, NOM_COUNT_TRACK
jbe DriftBelowNOMlea eax, [edi - NOM_COUNT_TRACK]
xor edx, edx
mov ecx, 210
div ecx
mov [ebp].GLDrift, eaxmov ecx, 210
mul ecx
shr eax, 4mov edi, eax
mov eax, [ebp].Drift
sub eax, NOM_COUNT_TRACK
sub eax, edimov edi, 0ffffh
sub edi, [ebp].GLDrift
inc edi
mov [ebp].GLDrift, edi

ret

DriftBelowNOM:

mov eax, NOM_COUNT_TRACK
sub eax, [ebp].Driftxor edx, edx
mov ecx, 210
div ecx
mov [ebp].GLDrift, eaxmov ecx, 210
mul ecx
shr eax, 4mov edi, NOM_COUNT_TRACK
sub edi, [ebp].Drift
sub edi, eaxmov eax, 0ffffh
sub eax, edi
inc eax
retCalculateDriftDelta endp
subttl -- Step --
page; *****
; BEGIN_MANUAL_ENTRY(Step, DPC/API/STEP)

; Name: / Step

; Description: Acquisition State Routine.

; On Entry: EAX N/A

; EBX

ECX Frame Data Space

N/A

N/A

EBP Adapter Data Space

ESI N/A

EDI N/A

Note: Interrupts are in any state.

EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

; Remarks: This routine is called by InitState.
; It can be called at process or interrupt time.

; See Also:

; END_MANUAL_ENTRY

; *****
; public Step
; procmov eax, [ebp].NextStepCount
inc eax
xor edx, edx
mov ecx, 4
div ecx
mov [ebp].NextStepCount, edxmov eax, [ebp].SearchLoc
cmp [ebp].SearchLocFound, FALSE
je DontUseNextStepor edx, edx
je StepSetGLOffset
cmp edx, 2
je StepSetGLOffsetinc eax
cmp edx, 1
je StepDivideBy3
inc eaxStepDivideBy3:
xor edx, edx
mov ecx, 3
div ecx
mov eax, edx

StepSetGLOffset:

mov eax, SearchTbl[eax * 4]
mov [ebp].GLOffset, eax
jmp StepCalcRx

DontUseNextStep:

mov ecx, SearchTbl[eax * 4]

```
mov    [ebp].GLOffset, ecx
inc    eax
xor    edx, edx
mov    ecx, 3
div    ecx
mov    [ebp].GLOffset, edx
```

StepCalcRx:

```
mov [ebp].Drift, 0
call CalculatorxFreq
mov [ebp].Drift, NOM_COUNT_TRACK
ret
```

```
Step      endp      subttl      -- InitState --
          page
```

```

; BEGIN_MANUAL_ENTRY( InitState, DPC/API/INITSTA )

```

```

; Name:
InitState

```

```

; Description: Acquisition State Routine.

```

; On Entry:	EAX	N/A	Data Space
;	EBX	Frame	Data Space
;	ECX	N/A	
;	EDX	N/A	
;	EBP	Adapter	Data Space
;	ESI	N/A	
;	EDI	N/A	

Note: Interrupts are in any state.

On Return:	EAX	Destroyed
;	EBX	Preserved
;	ECX	Destroyed
;	EDX	Destroyed
;	EBP	Preserved
;	ESI	Preserved
;	EDI	Preserved

Flags:

Note: Interrupts preserved.

```

; Remarks: This routine is called by DriverCallBack.
;          It can be called at process or interrupt time.

```

; See Also:

; END MANUAL ENTRY

```
public InitState
InitState proc
```

```
cmp [ebp].TrackingMode, TRUE
jne InitStateNotTracking
```

```
cmp     DebugMask, 0
je      InitStateNoMsg
```

```

mov    eax, offset InitStateTrackMsg
cmp    eax, LastDebugMessage
je      InitStateNomsg
mov    eax, LastDebugMessage, eax
push   eax
push   DPSCScreen
call   OutputToScreen
lea    esp, [esp + (2 * 4)]

```

InitStateNoMsq:

```

call    CalculatedDriftDelta
add     eax, NOM_COUNT_REACQ

mov     edx, [ebp].IOCountNomLowAddr
dx, al

out     dx, 8

shr     al, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

```

```
mov     edx, [ebp].IOGateCountHighAddr
mov     eax, [ebp].ReacqGateCount
out     dx, al
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     edx, [ebp].IOBtrControlAddr
xor     eax, eax
out     dx, ax
```

```
mov     edx, [ebp].IOAfcControlAddr
in      al, dx
or      al, SWP_ENA_MASK
out     dx, al
```

```

mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
out     dx, al

```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     InitStatesetMode
and     al, NOT MODE_MASK
jmp     InitStatesetCheckRate

```

```
InitStateSetMode:
or
InitStateCheckRate:
out
```

```

mov     edx, [ebp].IO SpareIOControlAddr
mov     al, 0ah
cmp     [ebp].ViterbiOnly, 2
je      InitStateSetRate
mov     al, 0bh
cmp     [ebp].ViterbiOnly, 1
je      InitStateSetRate

```

```

mov al, 0fh
InitStateSetRate:
out dx, al
mov [ebp].NextState, ENABLE_BTR
jmp InitStateCheckPointing

InitStateNotTracking:
cmp DebugMask, 0
je InitStateNOMsg
mov eax, offset InitStateNoTrackMsg
cmp eax, LastDebugMessage
je InitStateNOMsg
mov LastDebugMessage, eax
push eax
push DPCScreen
call OutputToScreen
lea esp, [esp + (2 * 4)]

InitStateNOMsg:
mov edx, [ebp].IOBitDetControlAddr
xor eax, eax
out dx, al

mov edx, [ebp].IOSpareIOControlAddr
mov al, 0ah
cmp [ebp].ViterbiOnly, 2
je InitStateNTSetRate
mov al, 0bh
cmp [ebp].ViterbiOnly, 1
je InitStateNTSetRate
mov al, 0fh
InitStateNTSetRate:
out dx, al

mov edx, [ebp].IODaAdOffsetControlAddr
xor eax, eax
out dx, al

mov edx, [ebp].IOAfcControlAddr
mov eax, [ebp].ModulationScheme
or eax, SWEEP_DIR_SENSE_MASK
out dx, al

mov edx, [ebp].IOSweepRateAddr
mov al, 8ah
out dx, al

mov edx, [ebp].IOGateCountHighAddr
mov eax, [ebp].ReceiveGateCount
out dx, al

mov edx, [ebp].IOCountDeltaAddr
mov eax, [ebp].SqrDeltaCount
out dx, al

mov eax, [ebp].NomCountSearch
[ebp].TuneCount, eax
mov edx, [ebp].IOCountNomLowAddr
out dx, al
shr eax, 8
mov edx, [ebp].IOCountNomHighAddr
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx

```

```

or al, RESET_FEC_ACQ_MASK
out dx, al

mov edx, [ebp].IOBtrControlAddr
xor eax, eax
out dx, al

mov edx, [ebp].IOAfcControlAddr
in al, dx
or al, SWP_ENA_MASK
out dx, al

mov edx, [ebp].IOAGcFirControlAddr
mov al, 10 OR AGC_SENSE_MASK
out dx, al

mov edx, [ebp].IOBtrControlAddr
in al, dx
or al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
out dx, al

mov edx, [ebp].IOCrkThrLowAddr
mov al, 60h
out dx, al

mov edx, [ebp].IOCrkAddr
mov al, 0e0h
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
al, SENA_MASK
cmp [ebp].ModulationScheme, BPSK
jne InitStateSetSena
or al, DEPUNC_BYPASS_MASK
InitStateSetSena:
out dx, al

mov edx, [ebp].IOCrkControlAddr
mov al, 16 OR CRK_GAIN_OFFSET OR CRK_DET_PWR_OFFSET
out dx, al

mov edx, [ebp].IOSynthSerControlAddr
in al, dx
cmp [ebp].ViterbiMode, LOWRATE
jne InitStateResetBtr

or al, RESET_BTR_ACC_MASK
and al, NOT MODE_MASK
jmp InitStateClearBtr
InitStateResetBtr:
or al, MODE_MASK OR RESET_BTR_ACC_MASK
out dx, al

in al, dx
and al, NOT RESET_BTR_ACC_MASK
out dx, al
call Step

mov [ebp].NextState, ACQ_PD

InitStateCheckPointing:
mov [ebp].RateCount, 0
mov [ebp].PointingFlag, TRUE

```

```

cmp [ebp].DemodCommand, POINTING_MODE
je InitStateExit
mov [ebp].PointingFlag, FALSE
InitStateExit:
mov [ebp].CurrentState, SYNTH_PRGM
mov [ebp].SignalQuality, 0
mov [ebp].DemodCommand, BUSY_MODE
mov [ebp].DemodStatus, UNLOCKED
mov [ebp].FecStatus, UNLOCKED
ret

```

```

InitState      endp
subttl  -- ProgTuner --
page

```

```

; *****
; BEGIN_MANUAL_ENTRY( ProgTuner, DPC/API/PROGTUN )
;
; Name: ProgTuner
; Description: Acquisition State Routine.
; On Entry:  EAX N/A
;            EBX Frame Data Space
;            ECX N/A
;            EDX N/A
;            EBP Adapter Data Space
;            ESI N/A
;            EDI N/A
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Preserved
;            EDI Preserved
;
; Flags:
;
; Note: Interrupts preserved.
;
; Remarks: This routine is called by Tune.
;          It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
; *****
;
; public ProgTuner
; proc
; EAX = data
; ECX = len
;
; dec ecx
; mov edx, 1
; shl edx, cl
; mov ecx, edx
; mov esi, eax
;
; ESI = Data
;
; *****
; BEGIN_MANUAL_ENTRY( Tune, DPC/API/TUNE )
;
; Name: Tune
; Description: Acquisition State Routine.
; On Entry:  EAX N/A
;            EBX Frame Data Space
;            ECX N/A
;            EDX N/A
;            EBP Adapter Data Space
;            ESI N/A
;            EDI N/A
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; *****

```

```

ProgTunerLoop:
jecxz ProgTunerExit
mov edx, [ebp].IOSynthSerControlAddr
in al, dx
test esi, ecx
je ProgTunerClear
or al, SDATA_MASK
and al, NOT SCLK_MASK
out dx, al
jmp ProgTunerDelay
ProgTunerClear:
and al, NOT (SCLK_MASK OR SDATA_MASK)
out dx, al
ProgTunerDelay:
shr ecx, 1
mov edx, [ebp].IOStatusAddr
in al, dx
in al, dx
mov [ebp].IOSynthSerControlAddr
in al, dx
or al, SCLK_MASK
out dx, al
mov edx, [ebp].IOStatusAddr
in al, dx
in al, dx
jmp ProgTunerLoop
ProgTunerExit:
mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SCLK_MASK
out dx, al
ret
ProgTuner      endp
subttl  -- Tune --
page
; *****
; BEGIN_MANUAL_ENTRY( Tune, DPC/API/TUNE )
;
; Name: Tune
; Description: Acquisition State Routine.
; On Entry:  EAX N/A
;            EBX Frame Data Space
;            ECX N/A
;            EDX N/A
;            EBP Adapter Data Space
;            ESI N/A
;            EDI N/A
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; *****

```

```

; EBX Preserved
; ECX Destroyed
; EDI Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
;
; Note: Interrupts preserved.
;
; Remarks: This routine is called by SynthPrmState.
; It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****
;
; public Tune
; proc
;
;     cmp [ebp].TunerTypeFound, SHARP
;     je TuneSharpPan
;     cmp [ebp].TunerTypeFound, PANASONIC
;     je TuneSharpPan
;     cmp [ebp].TunerTypeFound, SHARP_CUSTOM
;     je TuneSharpCustom
;     ret
;
; TuneSharpPan:
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     or al, SENA_MASK
;     out dx, al
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     and al, NOT (SCLK_MASK OR SDATA_MASK)
;     out dx, al
;
;     cmp [ebp].TrackingMode, 0
;     jne TuneSetNA
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     and al, NOT SENA_MASK
;     out dx, al
;
;     mov eax, 2ch
;     cmp [ebp].TunerTypeFound, SHARP
;     je TuneProgTuner
;     mov eax, 0ech
;
; TuneProgTuner:
;     mov ecx, 8
;     mov call ProgTuner
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     or al, SENA_MASK
;     out dx, al
;
;     mov edx, [ebp].IOStatusAddr
;     in al, dx
;     in al, dx

```

TuneSetNA:

```

;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     and al, NOT SENA_MASK
;     out dx, al
;
;     mov eax, [ebp].ChannelNumber
;     add eax, [ebp].GLDrift
;     xor edx, edx
;     mov ecx, SYNTH_RATIO
;     div ecx
;     mov edi, edx
;     or eax, 3000h
;
;     mov ecx, 16
;     call ProgTuner
;
;     mov eax, edi
;     mov ecx, 8
;     call ProgTuner
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     or al, SENA_MASK
;     out dx, al
;
;     ret

```

TuneSharpCustom:

```

;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     and al, NOT SENA_MASK
;     out dx, al
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     and al, NOT (SCLK_MASK OR SDATA_MASK)
;     out dx, al
;
;     mov eax, 50h OR 8001h
;     mov ecx, 16
;     call ProgTuner
;
;     mov edx, [ebp].IOSynthSerControlAddr
;     in al, dx
;     or al, SENA_MASK
;     out dx, al

```



```

mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

mov     eax, [ebp].ChannelNumber
add     eax, [ebp].GLDrift
xor     edx, edx
mov     ecx, SYNTH_RATIO
div     ecx, edi, edx

mov     ecx, 11
call    ProgTuner

mov     eax, edi
shl     eax, 1
mov     ecx, 9
call    ProgTuner

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, SENA_MASK
out     dx, al

mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

ret

Tune    endp
subttl  -- SynthPrmState --
page

; *****
; BEGIN_MANUAL_ENTRY( SynthPrmState, DPC/API/SYNTHPS )
;
; Name:      SynthPrmState
; Description: Acquisition State Routine.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are in any state.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed

```

```

; EDX  Destroyed
; EBP  Preserved
; ESI  Preserved
; EDI  Preserved
;
; Flags:
;
; Note:      Interrupts preserved.
;
; Remarks:   This routine is called by DriverCallBack.
;            It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
; *****
;
; public SynthPrmState
; SynthPrmState proc
;
;     cmp     DebugMask, 0
;     je      SynthPrmStateNoMsg
;     mov     eax, offset SynthPrmMsg
;     cmp     LastDebugMessage
;     je      SynthPrmStateNoMsg
;     mov     LastDebugMessage, eax
;     push    eax
;     push    DPCScreen
;     call    OutputToScreen
;     lea     esp, [esp + (2 * 4)]
;     SynthPrmStateNoMsg:
;     call    Tune
;
;     [ebp].TrackingMode, 0
;     [ebp].NextState, ACQ_PD
;     SynthPrmClearT2
;
;     [ebp].MaxSsf, 0
;     [ebp].SsfAvg, 0
;     [ebp].SsfWait, 0
;     mov     eax, [ebp].SsfCheckPoints
;     [ebp].MaxCount, eax
;     [ebp].T2Count, 60
;     [ebp].CurrentState, ACQ_PD_DELAY
;     ret
;
; SynthPrmClearT2:
;     mov     [ebp].T2Count, 0
;     [ebp].CurrentState, ACQ_PD_DELAY
;     ret
;
; SynthPrmState endp
; subttl  -- AcqPDDelayState --
; page
; *****
; BEGIN_MANUAL_ENTRY( AcqPDDelayState, DPC/API/ACOPDDS )
;
; Name:      AcqPDDelayState
; Description: Acquisition State Routine.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space

```



```

add     [ebp].SqfAvg, eax
cmp     eax, [ebp].MaxSqf
jbe     AcqPDDecMaxCount

mov     [ebp].MaxSqf, eax
mov     eax, [ebp].TuneCount
mov     [ebp].BestTuneCount, eax

AcqPDDecMaxCount:
dec     [ebp].MaxCount
jne     AcqPDMaxCountNotZero

mov     edx, [ebp].IOSweepRateAddr
mov     al, 8ah
out     dx, al

mov     edx, [ebp].IOCountNomLowAddr
mov     eax, [ebp].BestTuneCount
out     dx, al

shr     eax, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

mov     edx, [ebp].IOCountDeltaAddr
mov     eax, [ebp].SqfDeltaCount
shl     eax, 1
out     dx, al

mov     [ebp].SqfAvg
mov     ecx, [ebp].SqfCheckPoints
xor     edx, edx
div     ecx, 2
add     eax, 2
mov     [ebp].SqfAvg, eax

mov     [ebp].T2Count, 40
mov     [ebp].NextState, ENABLE_BTR
mov     [ebp].CurrentState, ACQ_PD_DELAY

AcqPDExit:
ret

```

```
AcqPDMMaxCountNotZero:
    mov     eax, [ebp].TuneCount
    add     eax, [ebp].SqfCheckStepSize
    mov     [ebp].TuneCount, eax

    mov     edx, [ebp].IOCountNomLowAddr
    out     dx, al

    mov     edx, [ebp].IOCountNomHighAddr
    shr     eax, 8
    out     dx, al

    mov     [ebp].T2Count, 20
    mov     [ebp].CurrentState, ACQ_PD_DELAY
    ret
```

```
AcqPDSState      endp
subttl -- EnableBTRState --
page
```

```

; BEGIN_MANUAL_ENTRY( EnableCTRState, DPC/API/ENBTRST )

```

Name:	EnableBTRState	
Description:	Acquisition State Routine.	
On Entry:	EAX N/A	
	EBX Frame Data Space	
	ECX N/A	
	EDX N/A	
	EBP Adapter Data Space	
	ESI N/A	
	EDI N/A	
Note:	Interrupts are in any state.	
On Return:	EAX Destroyed	
	EBX Preserved	
	ECX Destroyed	
	EDX Destroyed	
	EBP Preserved	
	ESI Preserved	
	EDI Preserved	
Flags:		
Note:	Interrupts preserved.	
Remarks:	This routine is called by DriverCallBack.	
	It can be called at process or interrupt time.	

```

; See Also:
;
;
; END_MANUAL_ENTRY
;
; *****
;
; public EnableBTRState
EnableBTRState proc
;
;
; *****
;

```

```

cmp     DebugMask, 0
je      EnableBTRStateNoMsg
mov     eax, offset EnableBTRMsg
cmp     eax, LastDebugMessage
je      EnableBTRStateNoMsg
mov     eax, LastDebugMessage, eax
push    eax
push    DPCScreen
push    OutputToScreen
call    esp, [esp + (2 * 4)]
lea     EnableBTRStateNoMsg:
mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, BTR_ERR_ENA_MASK
out     dx, al

mov     [ebp].CurrentState, START_SEARCH_FOR_FEC
ret

```

```

EnableBTRState      endp
:      subtbl      -- StartSearchForFECState --
:      page
:
: .....
:
: BEGIN MANUAL_ENTRY( StartSearchForFECState, DPC/API/SRCHFEC )
:
:

```



```

CheckForFEClockState proc
    cmp     DebugMask, 0
    je      CheckForFEClockStateNoMsg
    mov     eax, offset CheckForFEClockStateMsg
    cmp     eax, LastDebugMessage
    je      CheckForFEClockStateNoMsg
    push    LastDebugMessage, eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    CheckForFEClockStateNoMsg:
        mov     edx, [ebp].IOStatusAddr
        in     al, dx
        and    al, FEC_LOCK_MASK
        je      CheckFECNotLocked
        mov     [ebp].DemodStatus, LOCKED
        mov     [ebp].FecStatus, LOCKED
        mov     edx, [ebp].IOGateCountHighAddr
        xor    eax, eax
        out    dx, al
        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].ReacqDeltaCount
        out    dx, al
        mov     edx, [ebp].IOBtrControlAddr
        in     al, dx
        and    al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out    dx, al
        in     al, dx
        or     al, 5
        out    dx, al
        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 500
        mov     [ebp].T2Count, 100
        mov     [ebp].CurrentState, TRACKING
        ret

CheckFECNotLocked:
    cmp     [ebp].T1Count, 0
    jne     CheckFECHaveT1Count

    mov     edx, [ebp].IOStatusAddr
    in     al, dx
    test    al, CRL_LOCK_MASK
    jne     CheckFECSetOtherMode

    mov     [ebp].CurrentState, INIT
    ret

CheckFECSetOtherMode:
    mov     [ebp].CurrentState, SET_OTHER_MODE
    CheckFECExit:
        ret

CheckFECHaveT1Count:
    mov     edx, [ebp].IOStatusAddr
    in     al, dx

```

```

    test    al, CRL_LOCK_MASK
    jne     CheckFECExit
    mov     eax, [ebp].MaxSgf
    cmp     eax, [ebp].SgfAvg
    jbe     CheckFECExit
    sub     eax, 2
    mov     [ebp].MaxSgf, eax
    mov     edx, [ebp].IOctHAddr
    out    dx, al
    ret

CheckForFEClockState endp
    subttl -- SetOtherModeState --
    page

; *****
; BEGIN_MANUAL_ENTRY( SetOtherModeState, DPC/API/SETOTHER )
;
; Name: SetOtherModeState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
;           EBX Preserved
;           ECX Destroyed
;           EDX Destroyed
;           EBP Preserved
;           ESI Preserved
;           EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallBack.
;          It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; *****
; public SetOtherModeState
; SetOtherModeState proc
    cmp     DebugMask, 0
    je      SetOtherModeStateNoMsg
    mov     eax, offset SetOtherModeStateMsg
    cmp     eax, LastDebugMessage
    je      SetOtherModeStateNoMsg
    mov     LastDebugMessage, eax
    push    DPCScreen
    push    OutputToScreen
    call

```

```

lea esp, [esp + (2 * 4)]
SetOtherModeStateNoMsg:
    mov edx, [ebp].IOSynthSerControlAddr
    in al, dx
    cmp [ebp].ViterbiMode, LOWRATE
    jne SetOtherModeToLow
    or al, MODE_MASK
    out dx, al
    mov [ebp].ViterbiMode, HIGHRATE
    jmp SetOtherModeIncrRate
SetOtherModeToLow:
    and al, NOT MODE_MASK
    out dx, al
    mov [ebp].ViterbiMode, LOWRATE
SetOtherModeIncrRate:
    inc [ebp].RateCount
    cmp [ebp].RateCount, 1
    jg SetOtherModeExit
    mov edx, [ebp].IOSynthSerControlAddr
    in al, dx
    or al, RESET_FEC_ACQ_MASK
    out dx, al
    mov [ebp].TlCount, 180
    mov edx, [ebp].IOSynthSerControlAddr
    in al, dx
    and al, NOT RESET_FEC_ACQ_MASK
    out dx, al
    mov [ebp].CurrentState, CHECK_FOR_FEC_LOCK
    ret
SetOtherModeExit:
    mov [ebp].CurrentState, INIT
    ret
SetOtherModeState
    subttl -- ReadWord --
    page
; *****
; BEGIN_MANUAL_ENTRY( ReadWord, DPC/API/READWORD )
;
; Name: ReadWord
; Description: Acquisition State Routine.
; On Entry: EAX N/A
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
; Note: Interrupts are in any state.

```

```

; On Return: EAX Destroyed
;           EBX Preserved
;           ECX Destroyed
;           EDX Destroyed
;           EBP Preserved
;           ESI Preserved
;           EDI Preserved
;
; Flags:
;
; Note: Interrupts preserved.
; Remarks: This routine is called by TrackingState,
;           PointingAcquisitionState and PointingTrackingState
;           It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; *****
; public ReadWord
; ReadWord
; proc
;
; push ebx
; xor ebx, ebx
; mov edx, edi
; in al, dx
; mov bh, al
;
; mov ecx, 4
; ReadWordLoop:
; mov edx, esi
; in al, dx
; mov bl, al
;
; mov edx, edi
; in al, dx
; cmp al, bh
; je ReadWordExit
;
; mov bh, al
; dec ecx
; jne ReadWordLoop
;
; ReadWordExit:
; mov eax, ebx
; pop ebx
; ret
;
; ReadWord
; subttl -- TrackingState --
; page
; *****
; BEGIN_MANUAL_ENTRY( TrackingState, DPC/API/TRCKST )
;
; Name: TrackingState
; Description: Acquisition State Routine.

```

```

; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are in any state.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:      Interrupts preserved.
;
; Remarks:   This routine is called by DriverCallback
;            It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY

```

```

;*****
;
;
;*****/

```

```

public TrackingState
TrackingState proc

```

```

    cmp     DebugMask, 0
    je      TrackingStateNoMsg
    mov     eax, offset TrackingStateMsg
    cmp     eax, LastDebugMessage
    je      TrackingStateNoMsg
    mov     LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    TrackingStateNoMsg:

```

```

    mov     edx, [ebp].IOStatusAddr
    in      al, dx
    test    al, FEC_LOCK_MASK
    jne     TrackingStateReadQuality

```

```

    in      al, dx
    test    al, CRL_LOCK_MASK
    je      TrackingStateZero
    cmp     [ebp].T2Count, 0
    jne     TrackingStateT1

```

```

TrackingStateZero:

```

```

    mov     [ebp].TrackingMode, 0
    mov     [ebp].CurrentState, INIT
    ret

```

```

TrackingStateT1:

```

```

    mov     [ebp].T1Count, 1000
    ret

```

```

TrackingStateReadQuality:

```

```

    xor     eax, eax
    mov     edx, [ebp].IORelSsfAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax

```

```

    cmp     DebugMask, 0
    je      SignalStrengthNoMsg
    cmp     LastSignalStrength, 0
    jne     SignalStrengthNoMsg

```

```

    mov     LastSignalStrength, 1
    cmp     eax, 200
    jb      SignalStrengthNone
    sub     eax, 200
    shl     eax, 1
    add     eax, 60
    jmp     SignalStrengthPrint

```

```

SignalStrengthNone:

```

```

    xor     eax, eax
    SignalStrengthPrint:
    push    eax
    push    offset SignalStrengthMsg
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (3 * 4)]

```

```

SignalStrengthNoMsg:

```

```

    cmp     [ebp].T1Count, 0
    jne     TrackingStateExit
    mov     [ebp].T1Count, 1000
    mov     [ebp].TrackingMode, 1

```

```

    mov     edi, [ebp].IOtuningHighAddr
    mov     esi, [ebp].IOtuningLowAddr
    call    ReadWord
    mov     [ebp].Drift, eax

```

```

    mov     ecx, 2
    cmp     eax, NOM_COUNT_TRACK + OFFSET_THRESHOLD
    ja      TrackingStateLocFound
    mov     ecx, 0
    cmp     eax, NOM_COUNT_TRACK - OFFSET_THRESHOLD
    jb      TrackingStateLocFound

```

```

    mov     ecx, 1
    TrackingStateLocFound:

```

```

    mov     [ebp].SearchLoc, ecx
    mov     [ebp].SearchLocFound, TRUE
    TrackingStateExit:
    ret

```

```

TrackingState endp
    subttl -- PointingAcquisitionState --
    page

```

```

;*****
;
;*****\

```

```

; BEGIN_MANUAL_ENTRY( PointingAcquisitionState, DPC/API/PTACOST )

```

```

; Name:      PointingAcquisitionState
; Description: Acquisition State Routine.
; On Entry:  EAX  !/A

```

```

; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallBack.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY

```

```

public PointingAcquisitionState
PointingAcquisitionState proc

```

```

    cmp DebugMask, 0
    je PointingAcquisitionStateNoMsg
    mov eax, offset PointingAcqStateMsg
    cmp eax, LastDebugMessage
    je PointingAcquisitionStateNoMsg
    mov LastDebugMessage, eax
    push eax
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (2 * 4)]
    PointingAcquisitionStateNoMsg:

```

```

    mov edx, [ebp].IOStatusAddr
    in al, dx
    test al, SWEEPING_MASK
    je PointingNotSweeping

    mov esi, [ebp].IOTuningLowAddr
    mov edi, [ebp].IOTuningHighAddr
    call ReadWord
    shl eax, 4
    mov [ebp].Drift, eax

    mov [ebp].DemodStatus, LOCKED

    xor eax, eax
    mov edx, [ebp].IOGateCountHighAddr
    out dx, al

    mov edx, [ebp].IOBtrControlAddr
    in al, dx
    and al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
    out dx, al

```

```

    in al, dx
    or al, 5
    out dx, al

    mov [ebp].NextStepCount, 1
    mov [ebp].TiCount, 1000
    mov [ebp].SearchLocFound, FALSE
    mov [ebp].CurrentState, POINTING_TRACKING
    ret

```

```

PointingNotSweeping:
    mov eax, [ebp].MaxSsqf
    sub eax, 2
    mov [ebp].MaxSsqf, eax
    mov edx, [ebp].IOctHAddr
    out dx, al
    cmp [ebp].TiCount, 0
    jne PointingAcqExit

```

```

    mov [ebp].CurrentState, INIT
    PointingAcqExit:
    ret

```

```

    PointingAcquisitionState endp
    subttl -- PointingTrackingState --
    page

```

```

; BEGIN_MANUAL_ENTRY( PointingTrackingState, DPC/API/PTTRKST )

```

```

; Name: PointingTrackingState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.

```

```

; Remarks: This routine is called by DriverCallBack.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY

```



```

; ESI Preserved
; EDI Preserved
;
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallBack.
;           It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; *****
; public InitDemod
; proc
; mov [ebp].CurrentState, HALT
; mov [ebp].RxFreq, 0
; mov [ebp].ViterbiMode, 0
; mov [ebp].DemodCommand, HALT_MODE
; mov [ebp].SearchLoc, 1
; mov [ebp].Drift, 0
; mov [ebp].GLOffset, 0
; mov [ebp].TrackingMode, FALSE
;
; value = read_bits (STATUS_ADDR, TUNER_TYPE_MASK);
; xor eax, eax
; mov edx, [ebp].IOStatusAddr
; in al, dx
; and al, TUNER_TYPE_MASK
; mov cl, al
;
; value |= read_bits (UNIT_ID_ADDR, TUNER_TYPE_2_MASK);
; mov edx, [ebp].IOUnitIDAddr
; in al, dx
; and al, TUNER_TYPE_2_MASK
; or al, cl
;
; cmp al, SHARP
; jne InitDemodPanasonic
;
; cmp [ebp].DebugMask, 0
; je FillInTunerVars
;
; push eax
; push offset SharpTunerMsg
; push DPCScreen
; call OutputToScreen
; lea esp, [esp + (2 * 4)]
; pop eax
; jmp FillInTunerVars
;
InitDemodPanasonic:
; cmp al, PANASONIC
; jne InitDemodSharpCustom
;
; cmp [ebp].DebugMask, 0
; je FillInTunerVars
;
; push eax
; push offset PanasonicTunerMsg
; push DPCScreen
; call OutputToScreen
; lea esp, [esp + (2 * 4)]

```

```

; pop eax
; jmp FillInTunerVars
;
InitDemodSharpCustom:
; cmp al, SHARP_CUSTOM
; jne InitDemodExit
;
; cmp [ebp].DebugMask, 0
; je FillInTunerVars
;
; push eax
; push offset SharpCustomTunerMsg
; push DPCScreen
; call OutputToScreen
; lea esp, [esp + (2 * 4)]
; pop eax
;
FillInTunerVars:
; mov [ebp].TunerTypeFound, eax
; mov [ebp].ReacqGateCount, 0f0h
; mov [ebp].ReacqDeltaCount, 2
; mov [ebp].NonCountSearch, NOM_COUNT_REACQ - 75
; mov [ebp].SgfCheckPoints, 11
; mov [ebp].SgfCheckStepSize, 15
; mov [ebp].SgfDeltaCount, 8
;
InitDemodExit:
; ret
;
InitDemod
; subttl -- ApplyDelay --
; page
; *****
; BEGIN_MANUAL_ENTRY ( ApplyDelay, DPC/API/APPLYDEL )
;
; Name: ApplyDelay
; Description: Acquisition State Routine.
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
;           EBX Preserved
;           ECX Destroyed
;           EDX Destroyed
;           EBP Preserved
;           ESI Preserved
;           EDI Preserved
;
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallBack.
;           It can be called at process or interrupt time.
; See Also:

```

```

; END_MANUAL_ENTRY
; *****
; public ApplyDelay
; ApplyDelay
;
; Apply delay to T1 Counter
;
; cmp [ebp].TiCount, 0
; je ApplyDelayT2
; cmp [ebp].TiCount, eax
; jb ApplyDelayClearT1
; sub [ebp].TiCount, eax
; jmp ApplyDelayT2
;
; ApplyDelayClearT1:
; mov [ebp].TiCount, 0
;
; Apply delay to T2 Counter
;
; ApplyDelayT2:
; cmp [ebp].T2Count, 0
; je ApplyDelayExit
; cmp [ebp].T2Count, eax
; jb ApplyDelayClearT2
; sub [ebp].T2Count, eax
; jmp ApplyDelayExit
;
; ApplyDelayClearT2:
; mov [ebp].T2Count, 0
;
; ApplyDelayExit:
; ret
;
; ApplyDelay subttl -- CalculateRxFreq --
; page
; *****
; BEGIN_MANUAL_ENTRY( CalculateRxFreq, DPC/API/CALCRXFQ )
;
; Name: CalculateRxFreq
; Description: Acquisition State Routine.
;
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
;
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
;
; *****
; Note: Interrupts preserved.
;
; Remarks: This routine is called by DriverCallBack.
; It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
; *****
; public CalculateRxFreq
; CalculateRxFreq proc
;
; USHORT_T freq, total, new_total, results;
; sub esp, 2 * 4
;
; total = [esp + 0]
; results = [esp + 4]
; freq = esi
; new_total = edi
;
; freq = S.Rx_Freq - FREQ_BASE;
; freq += S.GL_offset;
; mov esi, [ebp].RxFreq
; sub esi, FREQ_BASE
; add esi, [ebp].GLOffset ; ESI = freq
;
; total = (freq * 2) / 729; ; EAX = freq
; mov eax, esi ; EAX = freq * 2
; shl eax, 1 ; EAX = freq * 2
; xor edx, ecx
; mov ecx, 729
; div ecx ; EAX = EAX / 729
; mov [esp + 0], eax ; total = EAX
;
; results = (total * 729) / 2;
; mov ecx, 729 ; EAX = total * 729
; mul ecx ; EAX = EAX * 2
; shr eax, 1 ; EAX = EAX / 2
; mov [esp + 4], eax ; results = eax
;
; freq = freq - results;
; sub esi, eax
;
; new_total = total * 10;
; mov eax, [esp + 0] ; EAX = total * 10
; mov ecx, 10 ; new_total = EAX
; mul ecx
; mov edi, eax
;
; total = (freq * 20) / 729;
; mov eax, esi ; EAX = freq
; mov ecx, 20 ; EAX = freq * 20
; mul ecx
; xor edx, ecx
; mov ecx, 729
; div ecx ; EAX = EAX / 729
; mov [esp + 0], eax ; total = EAX
;
; results = (total * 729) / 20;
; mov ecx, 729 ; EAX = total * 729
; mul ecx ; EAX = total * 729
; xor edx, ecx
; mov ecx, 20
; div ecx ; EAX = EAX / 20
;
; *****

```

```

mov     [esp + 4], eax
; results = EAX

; freq = freq - results;
sub     esi, eax

; new_total += total;
add     edi, [esp + 0]

; new_total *= 10;
mov     eax, edi
mov     ecx, 10
mul     ecx
mov     edi, eax

; total = (freq * 200) / 729;
mov     eax, esi
mov     ecx, 200
mul     ecx
xor     edx, edx
mov     ecx, 729
div     ecx
mov     [esp + 0], eax

; results = (total * 729) / 200;
mov     ecx, 729
mul     ecx
xor     edx, edx
mov     ecx, 200
div     ecx
mov     [esp + 4], eax
; results = EAX

; freq = freq - results;
sub     esi, eax

; new_total += total;
add     edi, [esp + 0]

; if (freq >= 2) new_total++;
cmp     esi, 2
jnb     CalcGetChannelNumber
inc     edi
; new_total++

CalcGetChannelNumber:
; S.Channel_Number = SYNTH_FIRST_CHANNEL + new_total;
add     edi, SYNTH_FIRST_CHANNEL
mov     [ebp].ChannelNumber, edi

cmp     DebugMask, 0
je      NoChannelMsg
push    edi
push    offset ChannelNumberMsg
push    DPCTScreen
call    OutputToScreen
lea     esp, [esp + (3 * 4)]
NoChannelMsg:

add     esp, 2 * 4
ret

CalculateRxFreq endp
subttl -- DriverCallBack --
page
; *****
; BEGIN_MANUAL_ENTRY( DriverCallBack, DPC/API/CALLBACK )

```

; Name: DriverCallBack

; Description: This routine will be executed once every second. It will detect if the hardware does not ack a transmission. If the hardware didn't ack then it will be reset, the transmission of that packet will be aborted and the next packet in the queue will be sent if there is one.

; On Entry: EAX N/A
 EBX @ Frame Data Space
 ECX N/A
 EDX N/A
 EBP @ Adapter Data Space
 ESI N/A
 EDI N/A

; Note: Interrupts are disabled.

; On Return: EAX Destroyed
 EBX Preserved
 ECX Destroyed
 EDX Destroyed
 EBP Preserved
 ESI Destroyed
 EDI Destroyed

; Flags:

; Note: Interrupts disabled.

; Remarks: This routine is called by the MSM.
 After this call returns, the MSM will schedule another call back.
 It is called at interrupt time.

; See Also: MSM\MSMCallBackProcedure

; END_MANUAL_ENTRY

```

; *****
; public DriverCallBack
; align 16
DriverCallBack proc
; cmp [ebp].TunerTypeFound, INVALID_TUNER
; jne DemodInitialized
; call InitDemod
; Check Rx Frequency
; DemodInitialized:
; cmp [ebp].RxFreq, 1330 * 10
; je CallBackCheckState
; mov [ebp].RxFreq, 1330 * 10
; cmp [ebp].RxFreq, 0
; jne CallBackCheckState
; mov eax, GlobalRxFreq
; RxFreq = GlobalRxFreq * 10
; mov ecx, 10
; mul ecx
; mov [ebp].PxFreq, eax
; *****

```

```

    cmp     [ebp].TrackingMode, FALSE
    je      CheckRxFreqSetMode
    call    CalculateRxFreq
    CheckRxFreqSetMode:
    mov     [ebp].DemodCommand, ACQUIRE_MODE

; Possibly set the CurrentState depending on DemodCommand
;
; CallBackCheckState:
    cmp     [ebp].DemodCommand, ACQUIRE_MODE
    je      CallBackSetInit
    cmp     [ebp].DemodCommand, POINTING_MODE
    jne     CallBackCheckHalt
    CallBackSetInit:
    mov     [ebp].CurrentState, INIT
    call    CallBackApplyDelay
    jmp     CallBackWatchDog

    CallBackCheckHalt:
    cmp     [ebp].DemodCommand, HALT_MODE
    jne     CallBackApplyDelay
    mov     [ebp].CurrentState, HALT

; Apply delay
;
    CallBackApplyDelay:
    mov     eax, 15
    call    ApplyDelay

    mov     eax, [ebp].CurrentState
    mov     esi, StateTbl[eax * 4]
    call    esi

    CallBackWatchDog:
    ret

    mov     eax, [ebp].BufferCount
    cmp     eax, [ebp].WatchBufferCount
    mov     [ebp].WatchBufferCount, eax
    jne     CallBackExit

    call    RefreshMipsStats

rFrames
    mov     eax, [ebp].MipsZeroAddrFrames

    cmp     eax, [ebp].WatchOldRejected
    mov     [ebp].WatchOldRejected, eax
    je      CallBackExit

    call    DriverISR

    mov     edx, [ebp].PicAddress
    in      al, dx
    SLOW
    or      eax, [ebp].PicMask
    out     dx, al
    SLOW
    in      al, dx
    SLOW
    and     eax, [ebp].PicUnMask
    out     dx, al

    CallBackExit:
    ret

```

```

; *****
; BEGIN_MANUAL_ENTRY( DriverSend, DPC/API/SEND )
;
; Name:      DriverSend
; Description: This routine will transfer the packet described in the
;              TCB to the NIC and initiate the send. TxStartTime and
;              RetryCounter must be set to enable the deadman timer.
;
; On Entry:   EAX      N/A
;              EBX      @ Frame Data Space
;              ECX      Padded Packet Length
;              EDX      N/A
;              EBP      @ Adapter Data Space
;              ESI      @ TCB
;              EDI      N/A
;
; Note:       Interrupts are disabled.
;
; On Return:  EAX      Destroyed
;              EBX      Preserved
;              ECX      Destroyed
;              EDX      Destroyed
;              EBP      Preserved
;              ESI      Destroyed
;              EDI      Destroyed
;
; Flags:
;
; Note:       Interrupts disabled.
;
; Remarks:    This routine is called by the MSM media module.
;              It is called at process or interrupt time.
;
; See Also:   ETHERTSM\EtherTSMDriverSend
;              ETHERTSM\MediaSendRaw8023
;              ETHERTSM\MediaSendEthernetII
;              ETHERTSM\MediaSend8022Over8023
;              ETHERTSM\MediaSend8022Snap
;
; END_MANUAL_ENTRY
; *****
;
; align 16
; DriverSend proc
;
;     lea     edi, [esi].TCBMediaHeader
;     cmp     word ptr [edi+12], 0608h
;     je      DriverSendArp
;
;     cmp     [ebp].AgentSendRoutine, 0
;     je      DriverSendExit
;
;     t back if we can't
;
;     push    ecx
;     ; Padded size
;     push    esi
;     ; Address of TCB
;     call    [ebp].AgentSendRoutine ; Give it to Slip Handler
;
;     ; Give i

```



```
subttl -- DriverISR --
page
```

```
*****
; BEGIN_MANUAL_ENTRY ( DriverISR, DPC/API/ISR )
;
```

```
; Name: DriverISR
```

```
; Description: This routine handles packet reception.
```

```
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP @ Adapter Data Space
; ESI N/A
; EDI N/A
```

```
; Note: Interrupts are disabled.
```

```
; On Return: EAX Destroyed
; EBX Destroyed
; ECX Destroyed
; EDX Destroyed
; EBP Destroyed
; ESI Destroyed
; EDI Destroyed
```

```
; Flags:
```

```
; Note: Interrupts disabled.
```

```
; Remarks: This routine is called by the MSM.
; It is called at interrupt time.
```

```
; See Also: MSM\MSMInterruptProcedure
```

```
; END_MANUAL_ENTRY
```

```
*****
```

```
align 16
public DriverISR
proc
```

```
; /* Set the adapters ram ptr to the next rbd to receive from */
; output(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
```

```
; DebugMessage DEBUG_ISR_ALL, ISREnterMsg
mov edx, [ebp].IOMsgRamPtr
mov eax, [ebp].CurrentAdapterRBD
shl eax, 1
add eax, RBD_BASE_ADDR
out dx, ax
```

```
; /* Keep processing packets until no more are left */
```

```
; /* NOTE: We are assuming that anyone looping back to DriverISRLoop
```

```
; * has set the MSR_RAM_PTR to the next RBD to examine.
```

```
; * while ((status = inport (bicd_base_addr + MSG_RAM)) & EMPTY)
```

```
; */
```

```
; DriverISRLoop:
xor eax, eax
; Clear upper status
```

```
mov edx, [ebp].IOMsgRam
in ax, dx
mov [ebp].IntStatus, eax

test eax, EMPTY
je DriverISRExit

inc [ebp].BufferCount
DebugMessage1 DEBUG_ISR, DebugRBDReceived, [ebp].CurrentAdapterRBD
```

```
; /* Jump if this is an error packet */
; if (status & 0x8F)
```

```
test [ebp].IntStatus, STATUS_ERROR
jne DriverISRBadPacket ; Any error bits set?
; Jump if not
```

```
; /* Heres a good packet. See if we have a buffer for it. */
; if (!global_pool[curr_rbd].buf_ptr)
```

```
mov esi, [ebp].CurrentECB
or esi, esi
jne DriverISRAddToECB ; Is this more of
; the last packet?
; Jump if it is.
```

```
if TIMESTAMP
; mov al, 'r'
; push eax
; call DPCtimestamp
; lea esp, [esp + 4]
; endif
```

```
mov esi, 1514
call MSMAllocatorECB
or eax, eax
jne DriverISRNoECB ; Max ECB size.
; Get an ECB
; Jump if no ECB.
```

```
; !!!! Satellite header is 12 bytes, EII is 14 bytes.
; !!!! Add 2 to offset to prevent double copy of turbo internet packets.
```

```
lea edi, [esi+RPacketEnvelope+2] ; EDI -> beginning
mov [esi].RPacketOffset, edi ; Store into ECB
mov [esi].RPacketSize, 0 ; Clear size
mov [ebp].CurrentECB, esi ; Store if split packet
jmp short DriverISRReadSize
```

```
; DriverISRAddToECB:
```

```
mov edi, [esi].RPacketOffset
add edi, [esi].RPacketSize
```

```
; DriverISRReadSize:
```

```
; /* ESI (curr_rbd) will be used a lot. Let's try to keep it intact. */
; /* Retrieve the length of the packet */
; length = inport(bicd_base_addr + MSG_RAM);
```

```
xor eax, eax
mov edx, [ebp].IOMsgRam
in ax, dx ; Clear upper word
; Msg Ram I/O port
; Get size of packet
```

```
add [esi].RPacketSize, eax ; Add to ECB size
```

```
DebugMessage1 DEBUG_ISR, DebugRBDSize, eax
```

```
; word_length = (length & 3) ? (length / 4) + 1 : (length/4);
```



```

mov     [ebp].LargestRx, eax

DebugRxAvail:
inc     [ebp].NumberLargerRx
add     eax, [ebp].TotalLargerRx
mov     edi, [ebp].NumberLargerRx
mov     [ebp].TotalLargerRx, eax
xor     edx, edx
div     edi
mov     [ebp].AveLargerRx, eax

DebugRxExit:
; ^^^ DEBUG
;
mov     edi, 1514
; Max Packet size
push    ecx
mov     al, 'R'
;
push    eax
call    DPCTimeStamp
;
lea     esp, [esp + 4]
pop     ecx
endif

xor     eax, eax
; Good packet

push    ebp
call    EthernetSMFastProcessGetRCB
pop     ebp
jne     DriverISRLoop
; no ecb returned
jmp     DriverISRDidntWantECB
hese

DriverISRNotOurs:
push    esi
call    eax
pop     esi
or      eax, eax
je      DriverISRLoop

DriverISRDidntWantECB:
MSMReturnRCB
jmp     DriverISRLoop

DriverISRFilterNext:
add     edi, size FilterStruct
lea     eax, [ebp].Filter[MAX_ADDR * size FilterStruct]
cmp     edi, eax
jbe     DriverISRFilterLoop
; Couldn't find filter address. Clean up.
;
MSMReturnRCB
DebugMessage6   DEBUG_ISR_ALL, FilterNone, [edx+0], [edx+1], [edx+2], [e
dx+3], [edx+4], [edx+5]
jmp     DriverISRLoop

DriverISRFilterSeqNoMatch:
inc     eax
inc     [edi].FilterSeqCount
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallerSR

```

```

DriverISRFilterNoFilterSeq:
inc     eax
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallerSR

```

```

DriverISRFilterNoPacketSeq:
mov     [edi].FilterSeqNum, 1
jmp     DriverISRFilterCallerSR

```

```

DriverISRFilterSkipRBDsLen:
MSMReturnRCB
DebugMessage2   DEBUG_ISR_ALL, FilterRBDLen, ecx, edx
jmp     DriverISRLoop

```

```

DriverISRNoECB:
DebugMessage   DEBUG_ISR, NoECBMsg
jmp     DriverISRBadNextRBD

```

```

DriverISRBadPacket:
mov     esi, [ebp].IntStatus
test    esi, FRAMING_ERR
je      DriverISRCheckAbort

```

```

DebugMessage   DEBUG_ISR, FramingErrMsg

```

```

DriverISRCheckAbort:
test    esi, ABORT
je      DriverISRCheckAlign

```

```

DebugMessage   DEBUG_ISR, AbortMsg

```

```

DriverISRCheckAlign:
test    esi, ALIGN_ERR
je      DriverISRCheckOverrun

```

```

DebugMessage   DEBUG_ISR, AlignErrMsg

```

```

DriverISRCheckOverrun:
test    esi, OVERRUN_ERR
je      DriverISRCheckDES

```

```

DebugMessage   DEBUG_ISR, OverrunErrMsg

```

```

DriverISRCheckDES:
test    esi, DES_ERR
je      DriverISRCheckCRC

```

```

DebugMessage   DEBUG_ISR, DESErrMsg

```

```

DriverISRCheckCRC:
test    esi, CRC_ERR
je      DriverISRErrorStats

```

```

DebugMessage   DEBUG_ISR, CRCErrMsg

```

```

DriverISRErrorStats:

```

```

DebugMessage   DEBUG_ISR, ReturnMsg

```

```

DriverISRBadNextRBD:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, [ebp].CurrentAdapterRBD
shl     eax, 1
add     ecx, RBD_BASE_ADDR
out     dx, ax

```

```
mov     edx, [ebp].IOMsgRam
xor     eax, eax
out     dx, ax

mov     eax, RBD_BUFFER_SIZE
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
inc     eax
cmp     eax, ADAP_RBD_NUM
jnb     DriverISRBadRBDWrap
xor     eax, eax
DriverISRBadRBDWrap:
mov     [ebp].CurrentAdapterRBD, eax

DebugMessage1 DEBUG_ISR_ALL, AdapterRBDMsg, eax
mov     edx, [ebp].IOMsgRamPtr
shl     eax, 1
add     eax, RBD_BASE_ADDR
out     dx, ax
jmp     DriverISRLoop

DriverISRExit:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, 0c3a0h
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
mov     edx, [ebp].IOMsgRam
out     dx, ax

DebugMessage1 DEBUG_ISR_ALL, ISRExitMsg, eax

mov     edx, [ebp].IOStatus
in      ax, dx
ret

DriverISR     endp
subttl  -- DriverDisableInterrupt --
page
; *****
; BEGIN_MANUAL_ENTRY( DriverDisableInterrupt, DPC/API/DISINT )
; Name:      DriverDisableInterrupt
; Description: This routine will disable the adapters ability to
;              interrupt the host.
; On Entry:  EAX  N/A
;            EBX  N/A
;            ECX  N/A
;            EDX  N/A
;            EBP  @ Adapter Data Space
;            ESI  N/A
;            EDI  N/A
; Note:      Interrupts are disabled.
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Preserved
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
; Flags:
; Note:      Interrupts disabled.
; Remarks:   This routine is called by the MSM.
; See Also:  DriverDisableInterrupt
; END_MANUAL_ENTRY
; *****
; align 16
DriverDisableInterrupt proc
xor     eax, eax
ret
DriverDisableInterrupt endp
subttl  -- DriverEnableInterrupt --
page
; *****
; BEGIN_MANUAL_ENTRY( DriverEnableInterrupt, DPC/API/ENINT )
; Name:      DriverEnableInterrupt
; Description: This routine will enable the adapters ability to
;              interrupt the host.
; On Entry:  EAX  N/A
;            EBX  N/A
;            ECX  N/A
;            EDX  N/A
;            EBP  @ Adapter Data Space
;            ESI  N/A
;            EDI  N/A
; Note:      Interrupts are disabled.
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Preserved
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
; Flags:
; Note:      Interrupts disabled.
; Remarks:   This routine is called by the MSM.
; See Also:  DriverDisableInterrupt
; END_MANUAL_ENTRY
; *****
; align 16
```

```
EBP  Preserved
ESI  Preserved
EDI  Preserved
```

Flags:

Note: Interrupts disabled.

Remarks: This routine is called by the MSM.

See Also: DriverEnableInterrupt

END_MANUAL_ENTRY

```
align 16
DriverDisableInterrupt proc
```

```
xor     eax, eax
ret
```

DriverDisableInterrupt endp

```
subttl  -- DriverEnableInterrupt --
page
```

BEGIN_MANUAL_ENTRY(DriverEnableInterrupt, DPC/API/ENINT)

Name: DriverEnableInterrupt

Description: This routine will enable the adapters ability to interrupt the host.

```
On Entry: EAX  N/A
          EBX  N/A
          ECX  N/A
          EDX  N/A
          EBP  @ Adapter Data Space
          ESI  N/A
          EDI  N/A
```

Note: Interrupts are disabled.

On Return:

```
EAX  Destroyed
EBX  Preserved
ECX  Preserved
EDX  Destroyed
EBP  Preserved
ESI  Preserved
EDI  Preserved
```

Flags:

Note: Interrupts disabled.

Remarks: This routine is called by the MSM.

See Also: DriverDisableInterrupt

END_MANUAL_ENTRY

```
align 16
```

```
DriverEnableInterrupt proc
```

```
ret
```

```
DriverEnableInterrupt endp
```

```
public DriverReset
```

```
subttl -- DriverReset --
```

```
page
```

```

; *****
; BEGIN_MANUAL_ENTRY( DriverReset, DPC/API/RESET )
;
; Name: DriverReset
; Description: This routine will reset and initialize the NIC.
; On Entry: EAX N/A
; EBX @ Frame Data Space
; ECX N/A
; EDX N/A
; EBP @ Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful (otherwise points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
; Flags:
; Note: Interrupts disabled.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMReset
; END_MANUAL_ENTRY
; *****

```

```
DriverReset proc near
```

```
inc [ebp].AdapterResetCount ; Increment stat counter.
```

```
xor eax, eax
```

```
ret
```

```
DriverReset endp
```

```
DefaultRxFrame proc
```

```
ret
```

```
DefaultRxFrame endp
```

```
extrn LSLGetStackIDFromName: near
```

```
ProtocolBindEvent proc
```

```

lea edx, IPName
call LSLGetStackIDFromName ; Return Stack ID in EBX
or eax, eax
jne short ProtocolBindExit

```

```

mov esi, [esp + Parm0]
cmp [esi+4], ebx ; IP Stack?
jne short ProtocolBindExit ; Nope
mov edx, [esi]
mov ebp, OurAdapterDataSpace ; EDX = Bound board number

```

```

xor ecx, ecx
ProtocolBindLoop:
mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
or ebx, ebx
jz ProtocolBindNext

```

```

cmp [ebx].MLIDBoardNumber, dx
jne ProtocolBindNext

```

```

mov eax, 1514
mov [ebx].MLIDMaximumSize, eax
sub eax, 14
mov [ebx].MLIDMaxRecvSize, eax
mov [ebx].MLIDRecvSize, eax

```

```
ProtocolBindNext:
```

```

inc ecx
cmp ecx, 4
jb ProtocolBindLoop
CPOP
ret

```

```
ProtocolBindEvent endp
```

```
ProtocolUnbindEvent proc
```

```
CPUsh
```

```

lea edx, IPName
call LSLGetStackIDFromName ; Return Stack ID in EBX
or eax, eax
jne short ProtocolUnbindExit

```

```

mov esi, [esp + Parm0]
cmp [esi+4], ebx ; IP Stack?
jne short ProtocolBindExit ; Nope
mov edx, [esi]
mov ebp, OurAdapterDataSpace ; EDX = Bound Board Number

```

```
xor ecx, ecx
```

```

ProtocolUnbindLoop:
mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
or ebx, ebx
jz ProtocolUnbindNext

```

```

cmp [ebx].MLIDBoardNumber, dx
jne ProtocolUnbindNext

```

```

mov eax, 1494
mov [ebx].MLIDMaximumSize, eax
sub eax, 14
mov [ebx].MLIDMaxRecvSize, eax

```

```
mov [ebx].MLIDRecvSize, eax
```

```
ProtocolUnbindNext:
```

```
inc ecx
cmp ecx, 4
jb ProtocolUnbindLoop
CPop
ret
```

```
ProtocolUnbindEvent      endp
```

```
subttl -- DriverInit --
page
```

```
*****
```

```
; BEGIN_MANUAL_ENTRY( DriverInit, DPC/API/INIT )
```

```
; Name: DriverInit
```

```
; Description: This routine will call EthernetsRegisterHSM,
; MSMRegisterParameters, MSMRegisterHardwareOptions,
; MSMSetHardwareInterrupt, MSMRegisterMLID, initialize
; variables in the Adapter Data Space and reset/initialize
; the card.
```

```
; On Entry:  EAX  N/A
;            EBX  N/A
;            ECX  N/A
;            EDX  N/A
;            EBP  N/A
;            ESI  N/A
;            EDI  N/A
```

```
; Note: Interrupts are enabled.
```

```
; On Return: EAX  0 if successful (otherwise it points to error message)
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by the OS at load time.
;           It is called at process time.
```

```
; See Also:  MSM\MSMRegisterParameters
;            MSM\MSMRegisterHardwareOptions
;            MSM\MSMSetHardwareInterrupts
;            MSM\MSMRegisterMLID
;            MSM\MSMScheduleInterruptCallBack
;            MSM\MSMScheduleAESCallBack
;            MSM\MSMEnablePolling
;            DriverReset
```

```
END_MANUAL_ENTRY
```

```
*****
```

```
DriverInit
```

```
CPush
```

```
if TIMESTAMP
```

```
lea eax, DPCTB
mov timestamp_begin, eax
mov timestamp_index, eax
add eax, TIMESTAMP_BUFFER_SIZE
mov timestamp_end, eax
```

```
endif
```

```
*****
; Fill in Driver Parameter Block fields.
; *****
```

```
mov DriverStackPointer, esp ; Fill in stack ->.
lea esi, DriverParameterBlock ; ESI -> Parm block.
call EthernetsRegisterHSM ; Get EBX.
jnz DriverInitError ; Jump if error.
```

```
; Yuck! We'll have to adjust the receive size down, since
; Hughes can't handle full 1500 byte packets with tunneling.
```

```
mov [ebx].MLIDMaximumSize, 1494
```

```
*****
; EBX -> Frame Data Space(Config Table).
; Let MSM Parse the command line.
; *****
```

```
*****
```

```
mov GlobalRxFreq, DEFAULT_RX_FREQ
```

```
mov eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRESS
```

```
SS
```

```
lea ecx, AdapterOptions
call MSMRegisterDriverParameters
jnz DriverInitError ; Jump if error.
```

```
*****
```

```
; Let MSM Register the hardware options.
```

```
*****
```

```
*****
```

```
call MSMRegisterHardwareOptions
```

```
cmp eax, 1 ; Error Registering?
ja DriverInitError ; Jump if so.
je DriverInitExit ; Skip if new frame.
```

```
mov OurAdapterDataSpace, ebp ; Save for later
mov DPCRxFrame, offset DefaultRxFrame
```

```
; Get a timer resource tag so that we can delay ourselves.
```

```
push TimerSignature
push offset TimerDesc
push DriverModuleHandle
call AllocateResourceTag
```



```

; = base + 10h + 1ah
add     ecx, 2
mov     [ebp].IODaAOffsetControlAddr, ecx
ddr = base + 10h + 1ch
add     ecx, 2
mov     [ebp].IOUnitIDAddr, ecx
10h + 1eh
add     ecx, 2

movzx   ecx, [ebx].MLIDIOPortsAndLengths
mov     al, 0bh
cmp     ecx, 100h
je      SetPort
dec     al
cmp     ecx, 140h
je      SetPort
dec     al
cmp     ecx, 180h
je      SetPort
dec     al
cmp     ecx, 1c0h
je      SetPort
dec     al
cmp     ecx, 200h
je      SetPort
dec     al
cmp     ecx, 240h
je      SetPort
dec     al
cmp     ecx, 280h
je      SetPort
dec     al
cmp     ecx, 2c0h
je      SetPort
dec     al
cmp     ecx, 300h
je      SetPort
dec     al
cmp     ecx, 340h
je      SetPort
dec     al
cmp     ecx, 380h
je      SetPort
dec     al
SetPort:
mov     dx, 279h
out     dx, al

; Lets Reset the adapter.
;
push     eax
mov     edx, [ebp].IOControl
mov     eax, CNTL_MRESET
out     dx, ax

mov     eax, [ebp].TimerTag
push     eax
push     2
call    DelayMyself
add     esp, (2 * 4)

mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax

```

```

; IO DA Offset Control A

```

```

; Unit ID Addr = base +

```

```

; AL = 0bh
; I/O port 100h?
; yes
; AL = 0ah
; I/O port 140h?
; yes
; AL = 9
; I/O port 180h?
; yes
; AL = 8
; I/O port 1c0h?
; yes
; AL = 7
; I/O port 200h?
; yes
; AL = 6
; I/O port 240h?
; yes
; AL = 5
; I/O port 280h?
; yes
; AL = 4
; I/O port 2c0h?
; yes
; AL = 3
; I/O port 300h?
; yes
; AL = 2
; I/O port 340h?
; yes
; AL = 1
; I/O port 380h?
; yes
; AL = 0

```

```

; Set Base I/O port

```

```

pop     eax
mov     dx, 279h
out     dx, al

```

```

; Make sure that we can set spare output

```

```

mov     edx, [ebp].IOControl
mov     eax, CNTL_SOUTPUT
out     dx, ax
; Set Spare Output

```

```

mov     edx, [ebp].IOStatus
in      ax, dx
test    eax, STAT_SOUTPUT
mov     eax, offset MsgIOSetFailed
je      DriverInitErrorReturn

```

```

; Make sure that we can clear spare output

```

```

mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax
; Clear Spare Output

```

```

mov     edx, [ebp].IOStatus
in      ax, dx
test    eax, STAT_SOUTPUT
mov     eax, offset MsgIOClearFailed
jne     DriverInitErrorReturn

```

```

; If no node override, default it.

```

```

cmp     dword ptr [ebx].MLIDNodeAddress, -1
jnz     short NodeIsSet
mov     [ebx].MLIDNodeAddress+0, 00h
mov     [ebx].MLIDNodeAddress+1, 80h
mov     [ebx].MLIDNodeAddress+2, 0aeh
mov     [ebx].MLIDNodeAddress+3, 00h
mov     [ebx].MLIDNodeAddress+4, 00h
mov     [ebx].MLIDNodeAddress+5, 01h

```

```

NodeIsSet:

```

```

; *****
; Download the MIPS code to the adapter.
; *****

```

```

mov     edx, [ebp].IOControl
mov     eax, CNTL_AUTO_INC
out     dx, ax
; Set to auto-increment mode

```

```

mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
; Set Adapter Ram Address
; to zero

```

```

mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax
; Set Adapter Ram Address
; to zero

```

```

mov     ecx, MipsCodeSize
mov     edx, [ebp].IOMsgRam
lea     esi, MipsCode
cld

```

```

CopyToAdapterLoop:

```

```

lodsb
out     dx, ax
; Get Mips Word
; Send it to adapter

```

```

dec      ecx
jnz      CopyToAdapterLoop

; Verify the download by reading the data back
;
mov      edx, [ebp].IOControl
mov      eax, CNTL_AUTO_INC
out      dx, ax

mov      edx, [ebp].IOMsgRamPtr
xor      eax, eax
out      dx, ax

mov      ecx, MipsCodeSize
mov      edx, [ebp].IOMsgRam
lea      esi, MipsCode
cld

VerifyAdapterLoop:
xor      eax, eax
lodsw
mov      edi, eax
in      ax, dx
cmp      edi, eax
lea      eax, MsgBadRAM
jne      DriverInitErrorReturn
dec      ecx
jnz      VerifyAdapterLoop

; *****
; Register our interrupt handler with the OS.
; *****
mov      edx, [ebp].IOStatus
in      ax, dx

; Set RBD base address
mov      edx, [ebp].IORbdBase
mov      eax, RBD_BASE_ADDR
out      dx, ax
mov      edx, [ebp].IOMsgRamPtr
out      dx, ax

mov      edx, [ebp].IORbdNum
mov      eax, ADAP_RBD_NUM
out      dx, ax
mov      ecx, eax

mov      edx, [ebp].IORbdBufLen
mov      eax, RBD_BUFFER_SIZE
out      dx, ax

mov      edx, [ebp].IOControl
mov      eax, CNTL_AUTO_INC
out      dx, ax

mov      edx, [ebp].IOMsgRam
xor      eax, eax
out      dx, ax

mov      eax, RBD_BUFFER_SIZE
out      dx, ax

SetupBuffersLoop:
mov      ecx, [ebp].IOMsgRam
xor      eax, eax
out      dx, ax

mov      eax, RBD_BUFFER_SIZE
out      dx, ax

dec      ecx
jnz      SetupBuffersLoop

; Enable the adapter.
movzx    ecx, [ebx].MLIDInterrupt
mov      esi, CNTL_IRQ3
mov      edx, 8
cmp      ecx, 8h
je      EnabledPC
mov      esi, CNTL_IRQ4
mov      edx, 10h
cmp      ecx, 4
je      EnabledPC
mov      esi, CNTL_IRQ5
mov      edx, 20h
cmp      ecx, 5
je      EnabledPC
mov      esi, CNTL_IRQ9
mov      edx, 2h
cmp      ecx, 8
je      EnabledPC
mov      esi, CNTL_IRQ10
mov      edx, 4h
cmp      ecx, 10
je      EnabledPC
mov      esi, CNTL_IRQ11
mov      edx, 8h
cmp      ecx, 11
je      EnabledPC
mov      esi, CNTL_IRQ12
mov      edx, 10h
cmp      ecx, 12
je      EnabledPC
mov      esi, CNTL_IRQ15
mov      edx, 80h
mov      ecx, 80h
mov      [ebp].PicMask, edx
not      edx
mov      [ebp].PicUnMask, edx
mov      [ebp].PicAddress, 21h
cmp      [ebx].MLIDInterrupt, 8
jnb      ClearOurInterrupt
mov      [ebp].PicAddress, 0a1h
ClearOurInterrupt:
mov      edx, [ebp].PicAddress
in      al, dx
and      eax, [ebp].PicUnMask
out      dx, al

mov      eax, esi
mov      edx, [ebp].IOControl
or      eax, CNTL_RX_EN OR CNTL_CPU_EN OR CNTL_INT_EN OR CNTL_SINGL_INT_O
R CNTL_AUTO_INC OR CNTL_SOUTPUT
out      dx, ax
mov      [ebp].IOEnableValue, eax

cmp      DebugMask, 0
je      OpenScreenExit
push     4e524353h
lea      eax, ScreenResourceName
push     eax
push     DriverModuleHandle
call     AllocateResourceTag
lea      esp, [esp + (3 * 4)]
or      eax, eax

```



```

je      OpenScreenExit
mov     ScreenRTag, eax

push    offset DPSCScreen
push    eax
push    offset ScreenName
call    OpenScreen
lea     esp, [esp + (3 * 4)]
or      eax, eax
jne     OpenScreenExit

push    offset NLName
push    0
push    DriverModuleHandle
call    GetNLNames
lea     esp, [esp + (3 * 4)]

push    0
push    offset Day
push    offset Month
push    offset Year
push    0
push    offset MinorVer
push    offset MajorVer
push    DriverModuleHandle
call    GetNLVersionInfo
lea     esp, [esp + (8 * 4)]

push    Year
push    Day
push    Month
push    MinorVer
push    MajorVer
push    offset NLName
push    offset DebugScreenHeader
push    DPSCScreen
push    OutputToScreen
call    lea     esp, [esp + (8 * 4)]

OpenScreenExit:
; *****
; Initialize RxControl and Filter entries.
; *****
;
lea     edi, [ebp].RxControl
mov     ecx, MAX_CHAN
xor     eax, eax
mov     edx, RBD_NOT_USED

InitRxControlLoop:
mov     [edi].RxChannel, edx
mov     [edi].RxESR, eax
add     edi, size RX_CNTL
dec     ecx
jne     InitRxControlLoop

lea     edi, [ebp].Filter
mov     ecx, MAX_ADPR

InitFilterLoop:
mov     [edi].FilterChannel, edx
mov     [edi].FilterTotalCount, eax
mov     [edi].FilterSeqCount, eax
mov     [edi].FilterSeqNum, eax

```

```

add     edi, size FilterStruct
dec     ecx
jne     InitFilterLoop
; *****
; Test Interrupts.
; *****
;
; Set ISR to test routine.
;
mov     [ebp].GotInterrupt, 0 ; Clear test flag.
mov     ecx, [ebp].ISRTag ; ECX = ISR resource tag.
push    0 ; No ExtraEOIFlag offset.
push    0 ; ShareFlag.
push    0 ; End of chain flag.
push    ecx ; ISR Resource tag.
lea     eax, TestDriverISR
push    eax
movzx   eax, [ebp].MLIDInterrupt ; ISR Entry point.
push    eax ; EAX = Interrupt number.
call    SetHardwareInterrupt ; Interrupt Number.
lea     esp, [esp + (6 * 4)] ; Set interrupt.
or      eax, eax ; Restore stack.
lea     eax, BadSRMsg ; Error setting interrupt?
jnz     DriverInitErrorReturn ; EAX -> Error Message.
; Exit if so.

mov     edx, [ebp].IOControl
mov     eax, [ebp].IOEnableValue
or      eax, CNTL_FORCEINT
out     dx, ax

sti
mov     eax, [ebp].TimerTag
push    eax
push    18
call    DelayMyself
add     esp, (2 * 4)
cli

movzx   eax, [ebp].MLIDInterrupt
lea     edx, TestDriverISR
push    edx
push    eax
call    ClearHardwareInterrupt ; Pass ISR Entry point.
lea     esp, [esp + (2 * 4)] ; Give interrupt back.
; Clean up stack.

mov     eax, [ebp].IOEnableValue
mov     dx, ax

lea     eax, NoInterruptMsg ; Error message.
cmp     [ebp].GotInterrupt, 0 ; Did we get it?
jpe     DriverInitErrorReturn ; Jump if not.
; *****
; EBX -> Frame Data Space(Config Table).
; EBP -> Adapter Data Space.
;
; Let MSM Set Hardware Interrupt.
; *****
; call    MSMSetHardwareInterrupt

```

```

; jnz DriverInitError ; Jump if error.
; *****
; Set TxFreeCount to make TSM happy.
; *****
mov [ebp].MSMTxFreeCount, 32 ; Allow 32 transmits simultaneously.
mov eax, 1 ; Schedule call back in 18 ticks
call SMScheduleIntTimeCallback ; Jump if error.
jnz DriverInitError
call DriverReset ; Initialize NIC.
jnz DriverInitErrorReturn ; Exit if error resetting.
mov [ebp].FirstTimeInit, 0 ; Disable DriverReset from testing the hardware again.
dec [ebp].AdapterResetCount ; Adjust reset count.
call MSMRegisterMLID ; Register MLID.
jnz DriverInitError ; Jump if error.
; Lets see if the adapter is locked up.
mov eax, [ebp].TimerTag
push eax
push 18
call DelayMyself
add esp, (2 * 4)
call RefreshMipsStats
test [ebp].MipsRxEnables, 8000000h ; This shouldn't be big
lea eax, LockedAdapterMsg
jne DriverInitErrorReturn
cmp DebugMask, 0
je DriverInitExit
movzx eax, [ebp].MLIDInterrupt
push eax
movzx eax, [ebp].MLIDIOPortsAndLengths
push eax
push offset DebugInitOK
push DPCScreen
call OutputToScreen
lea esp, [esp + (4 * 4)]
DriverInitExit:
mov [ebp].MLIDMaxRecvSize, 1400
xor eax, eax
CPop
ret

DriverInitErrorReturn:
push eax ; Save error message.
call MSMReturnDriverResources ; Return resources.
mov eax, DPCScreen
or eax, eax
je DriverInitErrorScreenClosed
push eax
call CloseScreen
lea esp, [esp + (1 * 4)]

; *****
; DriverInitError:
; esi, eax
; mov MSMPrintString
; call
; or
; CPop
; ret
; DriverInit
; subttl -- DriverShutdown --
; page
; *****
; BEGIN_MANUAL_ENTRY( DriverShutdown, DPC/API/SHUTDOWN )
; Name: DriverShutdown
; Description: This routine will turn off the NIC.
; On Entry: EAX N/A
; EBX @ Frame Data Space
; ECX 0 if Permanent Shutdown
; EDX N/A
; EBP @ Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful
; EBX Preserved
; ECX Preserved
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMShutdown
; END_MANUAL_ENTRY
; *****
DriverShutdown proc
or ecx, ecx
jne DriverShutdownAdapter
mov eax, [ebp].AgentRemoveRoutine
or eax, eax
je DriverShutdownAdapter
call eax

```

```
mov [ebp].AgentRemoveRoutine, 0
```

```
DriverShutdownAdapter:
```

```
pushfd
cli
mov edx, [ebp].IOControl
xor eax, eax
out dx, ax
mov edx, [ebp].IOStatus
in ax, dx
```

```
or ecx, ecx
jne DriverShutdownExit
mov edx, [ebp].IOControl
mov eax, CNTL_MRESET
out dx, ax
```

```
mov eax, DPCScreen
or eax, eax
je DriverShutdownScreenClosed
push eax
call CloseScreen
lea esp, [esp + (1 * 4)]
mov DPCScreen, 0
DriverShutdownScreenClosed:
```

```
mov eax, [ebp].ProtocolBindID
or eax, eax
je DriverShutdownExit
push eax
call UnRegisterEventNotification
add esp, (1 * 4)
; Pass Event ID.
; Unregister event.
; Clean up stack.

mov eax, [ebp].ProtocolUnbindID
or eax, eax
je DriverShutdownExit
push eax
call UnRegisterEventNotification
add esp, (1 * 4)
; Pass Event ID.
; Unregister event.
; Clean up stack.
```

```
DriverShutdownExit:
popfd
xor eax, eax
ret
; Good Return code.
```

```
DriverShutdown endp
subttl -- DriverRemove --
page
```

```
; BEGIN_MANUAL_ENTRY( DriverRemove, DPC/API/REMOVE )
```

```
; Name: DriverRemove
```

```
; Description: This routine call the MSM to return our resources.
```

```
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP N/A
; ESI N/A
```

```
N/A
```

```
Note: Interrupts are in any state.
```

```
On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
```

```
Flags:
```

```
Note: Interrupts preserved.
```

```
Remarks: This routine is called by the OS at unload.
; It is called at process time.
```

```
See Also: MSM\MSMDriverRemove
```

```
END_MANUAL_ENTRY
```

```
DriverRemove proc
```

```
CPush
mov eax, DriverModuleHandle
call MSMDriverRemove
CPop
ret
```

```
DriverRemove endp
```

```
OSCODE ends
```

```
end
```

```
/* interface between Helius DPCNE and Hughes DPCPE */
```

```
extern "C" {
#include <nwsemaph.h>
#include "sys_win.hhi"
#include "dpcutils.h"
#include "dbsinwin.h"
#undef VIRTUAL
#include "dpcagent.h"
#include <assert.h>
}
```

```
int PD_ESR(ECB*);
void DloHangup(void);
void DPCPDterminate(void);
void DPCPDbackground(void);
void DPCFileMain(void* arg); // thread
```

```
#include "sfwatch.h"
#include "sfqview.h"
#include "sfxparsr.h"
```

```
unsigned long GetTickCount(void) {
return clock() * 1000 / CLOCKS_PER_SEC;
}
```

```
extern int DloState;
extern LONG DloPxmmitCount;
extern LONG DloPxmmitBufferSize;
extern LONG DloRcvCount;
extern LONG DloConn;
```

```
int DloGetCurrentState(void) {
if 1
return (DloState == DLOS_CONN && DloConn == DLO_CONN_PACKAGE) ? DLOS_CONN : DL
OS_IDLE;
#else
return DloState;
#endif
}
```

```
int DloPortEmpty(void) {
if 1
return DloPxmmitCount == 0;
#else
return DloAndCommEmpty();
#endif
}
```

```
int DloPortOpen(void) {
return AIOPortHandle != (-1);
}
```

```
int DloGetStatus(tDloStatus* pStatus) {
if (pStatus == 0)
return (-1);
if (!DloPortOpen())
return (-1);
pStatus->iState = DloGetCurrentState();
pStatus->iXmitBytesBuffered = DloPxmmitCount;
pStatus->iXmitBufferSize = DloPxmmitBufferSize;
pStatus->iRcvBytesBuffered = DloRcvCount;
pStatus->iRcvBufferSize = DLOBUF3IZE;
return 0;
}
```

```
int DloGetBufSize(void) {
```

```
return DloPxmmitBufferSize - DloPxmmitCount;
}
```

```
DWORD DloExtendInactivityTimer(long) {
}
```

```
void DloHangup(void) {
}
```

```
void DloDispatch(void) {
}
```

```
/*
* Returns whether the adapter can gain access to the passed group ID
* The group ID includes a version number.
*/
```

```
long DLLAPI CDBCheckGroupID(CdbCfg_t *cfg)
{
```

```
if (find_pacau(cfg->groupid, cfg->ver) != NULL)
return(CAS_IMPLICIT);
if (find_dacau(cfg->groupid, cfg->ver) != NULL)
return(CAS_AUTHENTICATED);
if (find_eacu(cfg->groupid, cfg->ver) != NULL)
return(CAS_EXPLICIT);
return(CAS_ERROR);
}
```

```
/*
* Returns a version number which increments when there have been
* ANY changes to the adapter's conditional access.
*/
```

```
long DLLAPI CDBCheckCACHange(void)
{
return CDBVersion;
}
```

```
struct PID {
PID() { DPCFilePID = GetThreadID(); }
~PID() { DPCFilePID = 0; }
};
```

```
struct Semaphore {
LONG handle;
```

```
Semaphore(long initial = 0) { handle = OpenLocalSemaphore(initial); }
~Semaphore() { if (handle) CloseLocalSemaphore(handle); }
void Signal(void) { SignalLocalSemaphore(handle); }
LONG Wait(int milliseconds = (-1)) { return TimedWaitOnLocalSemaphore(handle,
(LONG)milliseconds); }
LONG value(void) { return ExamineLocalSemaphore(handle); }
LONG operator --(void);
};
```

```
inline LONG Semaphore::operator --(void) {
LONG v = value();
if (v == 0)
return v;
WaitOnLocalSemaphore(handle);
return v - 1;
}
```

```
Semaphore* DPCPDSemaphore;
SfxDispatcher* pDispatcher;
QUEUEVIEWER* pQueueViewer;
ECBQueue DPCPDQueue;
```

```

int PD_ESR(ECB* ecb) {
    Enqueue_IntsDisabled(&DPCPDQueue, ecb);
    return 0;
}

long BicddSignText(char* p_string,
                    unsigned long size,
                    char* p_sign) {
    return DIOSignText(p_string, size, p_sign);
}

long BicddGetSN(char* p_serial_num) {
    DIOSign(p_serial_num);
    return 0;
}

long BicddOpenChannel(BICDD_CHANNEL_CONFIG* channel_config) {
    if (channel_config->num_addresses != 1)
        return 1;

    DPCPDSemaphore = new Semaphore();
    if (DPCPDSemaphore->handle == 0) {
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        return 2;
    }
    DPCPDQueue.semaphore = DPCPDSemaphore->handle;

    // there is actually an overflow here IRT channel being a short!
    long ret = DIOOpenChannel(channel_config->address[0],
                              PD_ESR,
                              (LONG*)&channel_config->channel);

    if (ret) {
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        DPCPDQueue.semaphore = 0;
        return ret;
    }

    long BicddCloseChannel(unsigned long channel) {
        long ret = DIOCloseChannel(channel);
        if (ret)
            return ret;
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        DPCPDQueue.semaphore = 0;
        while (DPCPDQueue.head) {
            ECB* ecb = Dequeue(&DPCPDQueue);
            CLSRReturnRcvECB(ecb);
        }
        return 0;
    }

    /*****
    *
    * ELEMENTS SECTION
    * ( Elements Table support )
    *
    *****/

    CDBelement_t Elements[MAXELEMENTS];

    static find_element_by_mac(MACAddr_t mac) {

```

```

int k;
int ret = -1;

```

```

for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
        memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
        ret = k;
        break;
    }
    return ret;
}

```

```

static add_element(unsigned long channel, ID id, unsigned char ver,
                  MACAddr_t mac, char pack_feed)
{
    int k, ret = CAS_OK;

```

```

    if(find_element_by_mac(mac) != -1)
        return(CAS_DUPLICATE_ADDR);
    for(k = 0; k < MAXELEMENTS; k++)
        if(Elements[k].in_use != 'Y')
            break;
    if(k == MAXELEMENTS)
        ret = CAS_ERROR;
    else {
        Elements[k].channel = channel;
        Elements[k].e_ver = ver;
        memcpy(&Elements[k].e_id, &id, sizeof(id));
        memcpy(&Elements[k].e_mac, &mac, sizeof(mac));
        Elements[k].in_use = 'Y';
        Elements[k].packfeed = pack_feed;
    }
    return ret;
}

```

```

static find_element_id(ID id, unsigned char ver)
{
    int k;
    int ret = -1;

```

```

    for(k = 0; k < MAXELEMENTS; k++) {
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_id, &id, sizeof(id)) == 0 &&
            Elements[k].e_ver == ver) {
            ret = k;
            break;
        }
        return ret;
    }
}

```

```

static del_element_by_mac(MACAddr_t mac)
{
    int k, ret = CAS_ERROR;

```

```

    for(k = 0; k < MAXELEMENTS; k++) {
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
            ret = CAS_OK;
            Elements[k].in_use = 'N';
            break;
        }
        return ret;
    }
}

```

```

/*****
 * Add / Delete Package Delivery Address
 */
/
 * Allows an application to request resection of a single additional DPC MAC
 * address. Caller supplies the address's elementID and version number and the
 * element's group ID and version number. CDB looks up the group key and
 * element key for the address and attempts to add the address via a
 * driver call
 */
long BicddAddPkgAddr(CdbCfg_t* cfg) {
    char e_id_txt[7];
    MUXpacau_t* pacau;
    MUXdacau_t* dacau;

    make_element_id((BYTE*)&cfg->elementid, e_id_txt);
    MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);

    dacau = find_dacau(cfg->groupid, cfg->ver);
    pacau = dacau ? (MUXpacau_t*)dacau : find_pacau(cfg->groupid, cfg->ver);
    if (pacau == NULL)
        return CAS_ERROR;
    if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
        return CAS_ERROR;
    if (DIOAddGroupAddress(cfg->channel,
        (BYTE*)&cfg->mac,
        (BYTE*)&pacau->g_key) ==
        ESUCCESS)
        return CAS_OK;
    del_element_by_mac(cfg->mac);
    return CAS_ERROR;
}

/
 * For use by package delivery. Allows an application to request
 * reception of a for-sale package ( a package from an explicit group).
 * Package delivery passes address to be received (including the version number)
 * plus the group key to be used to receive the package. This group key was
 * received via explicit request transaction with the NOC.
 * CDB creates the corresponding element key and calls WBicddAddress.
 */
long BicddAddExpAddr(CdbCfg_t* cfg) {
    if (find_eacau(cfg->groupid, cfg->ver) == 0)
        return CAS_ERROR;

    char e_id_txt[7];
    make_element_id((BYTE*)&cfg->elementid, e_id_txt);
    MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);
    if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
        return CAS_ERROR;
    if (DIOAddGroupAddress(cfg->channel,
        (BYTE*)&cfg->mac,
        (BYTE*)&cfg->expl_g_key) == ESUCCESS)
        return CAS_OK;
    del_element_by_mac(cfg->mac);
    return CAS_ERROR;
}

/
 * Allows the application to discontinue reception of a single DPC MAC
 * Package Delivery supplies the element id and version number. CDB merely
 * reformats these values into a DPC MAC address and relays it to WINBICDD.
 */

```

```

long BicddDeletePkgAddr(CdbCfg_t* cfg) {
    int element = find_element_id(cfg->elementid, cfg->ver);
    if (element == (-1))
        return CAS_ERROR;
    if (DIODeleteAddress(cfg->channel, (BYTE*)&Elements[element].e_mac))
        return CAS_ERROR;
    del_element_by_mac(Elements[element].e_mac);
    return CAS_OK;
}

long BicddPoll(unsigned long channel) {
    return DPCDPSemaphore ? DPCDPSemaphore->value() : (-1);
}

long BicddReceive(unsigned long channel,
    BICDD_BUFFER* p_buffers,
    unsigned long buf_size,
    long timeout) {
    if (DPCDPSemaphore == 0)
        return (-1);
    if (DPCPDQueue.head == 0 && DPCDPSemaphore->Wait(timeout) != 0)
        return 0;
    ECB* ecb = DPCPDQueue.head;
    int r = 0;
    int n = buf_size / sizeof(BICDD_BUFFER);
    for (; ecb && n > 0; ecb = ecb->ECB_NextLink, --n) {
        p_buffers->data_size = ecb->ECB_Fragment[0].FragmentLength;
        p_buffers->buf_ptr = ecb->ECB_Fragment[0].FragmentAddress;
        p_buffers->last = 1;
        ++p_buffers;
        ++r;
    }
    return r * sizeof(BICDD_BUFFER);
}

long BicddFreeBuffers(unsigned long channel,
    BICDD_BUFFER* p_buffers,
    unsigned long buf_size) {
    int n = buf_size / sizeof(BICDD_BUFFER);
    int i = min(n, DPCDPSemaphore->value());
    for (; n > 0 && DPCPDQueue.head; --n)
        CLSLReturnRcvECB(DepQueue(&DPCPDQueue));
    for (; i > 0; --i)
        --*DPCDPSemaphore;
    return n;
}

long BicddGetSiteID(char* buffer) {
    if (!SiteID)
        return (-1);
    strncpy(buffer, (char*)SiteID, 9);
    return 0;
}

BOOL BicddGetSatelliteStatus(BICDD_SAT_STATS* Stats, long chan) {
    Stats->MarginalCutoff = MARGINAL_ACO_VALUE;
    Stats->NormalCutoff = NORMAL_ACO_VALUE;
    Stats->CurrentValue = DPCGetSignalStrength();
    return TRUE;
}

// The name of the registry/ini key values accessed in this module
static char* PREGKEY_DeleteOnDelivery = "DeleteOnDelivery";
static char* PREGKEY_CooperativeLoading = "CooperativeLoading";
static char* PREGKEY_RebuildOnStartup = "RebuildOnStartup";

```

```
static char* pREGKEY_Reconcile = "Reconcile";
static char* pREGKEY_EnableDebug = "EnabledDebug";
static char DBS_NAME[] = __FILE__;
static const char magic_key[] = {
    0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11
};
```

```
/* *****
```

```
EXPORTED FUNCTION
```

```
DPCCancelDownload(LONG fileID)
```

```
Description:
```

```
This routine cancels the download of the file associated
with the fileID if one is pending. This means closing
any open file handles and stopping the modem thread.
```

```
Input:
```

```
fileID - File ID of file to cancel
```

```
Output:
```

```
nothing
```

```
Returns:
```

```
0 if download was canceled
```

```
*****
```

```
static struct {
```

```
LONG control;
```

```
LONG ret;
```

```
LONG fileID;
```

```
BOOL cancel;
```

```
} crossover;
```

```
LONG DPCCancelDownload(LONG fileID)
```

```
{
    while (crossover.control)
        delay(100);
    crossover.fileID = fileID;
    crossover.cancel = TRUE;
    crossover.control = GetThreadID();
    while (crossover.control)
        delay(100);
```

```
/* Force the help package status to idle */
UpdateHelpPortal();
```

```
return crossover.ret;
```

```
LONG DPCDownloadAFire(LONG fileID)
```

```
{
    while (crossover.control)
        delay(100);
    crossover.fileID = fileID;
    crossover.cancel = FALSE;
    crossover.control = GetThreadID();
    while (crossover.control)
        delay(100);
```

```
return crossover.ret;
```

```
if (!pDispatcherLro) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcherLro");
    goto cleanup;
}
```

```
void DPCPDTerminate(void) {
    pDispatcher->Terminate();
}
```

```
void DPCPDBackground(void) {
    PDI_FillList_cross();
}
```

```
if (crossover.control) {
    LONG fileID = crossover.fileID;
    if (fileID != fsm.getFileid() && fsm.unique(fileID) != SFX_OK)
        crossover.ret = (LONG)-1;
    else if (crossover.cancel) {
        crossover.ret = fsm.dispatch(fileID, SFXFSM_FILE_NOT_WANTED);
        pDispatcher->CancelLoadingFileID(fileID);
    }
}
```

```
else if (fsm.isRequestable(fileID)) {
    fsm.dispatch(fileID, SFXFSM_PRECOMMIT);
    crossover.ret =
        fsm.dispatch(fileID,
```

```
fsm.isForSale(fileID) ? SFXFSM_PURCHASE : SFXFSM_FILE_WANTED);
}
```

```
sendret:
    ResumeThread(crossover.control);
    crossover.control = 0;
}
```

```
// this is adapted from sfxdemp.cpp WinMain and dpcfile.c DPCFileMain
```

```
void DPCFileMain(void* arg) {
    PID pid;

    if (!PackageDelivery)
        return;

    DbsProcInit("DPCPD");
```

```
if ((arg && strcmp((char*)arg, "rebuild") == ESUCCESS) ||
    DPCGetProfileInt(PROF_PACKAGEDDELIVERY, pREGKEY_RebuildOnStartup, 0)) {
    DPCSetProfileInt(PROF_PACKAGEDDELIVERY, pREGKEY_RebuildOnStartup, 0);
```

```
int n = fsm.Rebuild();
if (n >= 0) {
    DBS_SEND_TRACE1(0, "File database rebuilt with %d entries restored", n);
}
```

```
else {
    DBS_SEND_TRACE("File database rebuild failed");
}
```

```
return;
```

```
// wait until DIOBoard initialized
while (DIOBoard == 0) {
    if (ExitingFlag)
        return;
    delay(500);
}
```

```
pDispatcherLro = new SfxDispatcherLro();
if (!pDispatcherLro) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcherLro");
    goto cleanup;
}
```

```
)
pDispatcher = new SfxDispatcher();
if (!pDispatcher) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcher");
    goto cleanup;
}
pQueueViewer = new QUEUEVIEWER(PDI_UpdatedDisplay);
if (!pQueueViewer) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct QUEUEVIEWER");
    goto cleanup;
}

if (DPCGetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_Reconcile, 1))
    fsm.ReconcileWith(frd);
cdb.rebuildDB();

while (!ExitingFlag) {
    pDispatcher->Run();
    DPCPDBackground();
}

pDispatcher->Stop(5000);

cleanup:
delete pQueueViewer;
delete pDispatcher;
delete pDispatcherLro;
```



```

#include "dpcagent.h"          /* Our header file */

/* "fix" conflicting types */
#define AllocateResourceTag __AllocateResourceTag__
#include <advanced.h>
#undef AllocateResourceTag
#include <nwbitops.h>

#define milliclock()          (GetHighResolutionTimer() / 10)

#define activityTimer        ECB_DriverWorkspace.Dws_i32val

/* various flags that control the filter */
#undef FILTER_DATA_ON_RST
#define WIDEN_TCP_WINDOW
#undef TCP_ACK_LATENCY /* 10 */
/* if used at all, define *only* 1 of the following */
#undef DPCinetMaxQueuedBytes /* 4096 */
#define DPCinetMaxQueuedBytes 64
/* if defined(DPCinetMaxQueuedBytes) && defined(DPCinetMaxQueuedPackets)
#error Only 1 of DPCinetMaxQueuedBytes and DPCinetMaxQueuedPackets allowed
#endif

/* various flags that control the tunnel */
#undef TUNNEL_ONLY_TCP

#define IP_VERS(x)            (((BYTE*)x)[0] >> 4)
#define IP_HD_LEN(x)         (((BYTE*)x)[0] & 0x0f)
#define IP_TOS(x)            (((BYTE*)x)[1])
#define IP_TOT_LEN(x)        (((WORD*)x)[1])
#define IP_FLAG_FRAG(x)      (((WORD*)x)[3])
#define IP_PROTO(x)          (((BYTE*)x)[9])
#define IP_CSUM(x)           (((WORD*)x)[5])
#define IP_SRC_ADDR(x)       (((LONG*)x)[3])
#define IP_DST_ADDR(x)       (((LONG*)x)[4])

#define IPPROTO_IPENCAP      0x04

#define UDP_SRC_PORT(x)      (((WORD*)x)[0])
#define UDP_DST_PORT(x)      (((WORD*)x)[1])

#define TCP_SRC_PORT(x)      (((WORD*)x)[0])
#define TCP_DST_PORT(x)      (((WORD*)x)[1])
#define TCP_ACKNUM(x)        (((LONG*)x)[2])
#define TCP_CODE(x)          (((BYTE*)x)[13])
#define TCP_WINDOW(x)        (((WORD*)x)[7])
#define TCP_CSUM(x)          (((WORD*)x)[8])

#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20

ECBQueue TxQ;
ECBQueue NewQ;

struct ResourceTagStructure* TxChainRTag = 0;
struct ResourceTagStructure* TxECBRRTag = 0;
LONG TxChainID;
struct ResourceTagStructure* RxChainRTag = 0;
struct ResourceTagStructure* RxECBRRTag = 0;
LONG RxChainID;
LONG DPC_IP_Address = 0;
static BYTE ConnectionMask[65536 / 8];

```

```

void RemoveECBQueue* q, ECB* ecb) {
    disable();
    if (ecb->ECB_NextLink)
        ecb->ECB_NextLink->ECB_PreviousLink = ecb->ECB_PreviousLink;
    else
        q->tail = ecb->ECB_PreviousLink;
    if (ecb->ECB_PreviousLink)
        ecb->ECB_PreviousLink->ECB_NextLink = ecb->ECB_NextLink;
    else
        q->head = ecb->ECB_NextLink;
    ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
}

LONG InetQueuePacket(ECB* ecb, LONG board, void* chainID) {
    board = board;
    chainID = chainID;
    /* not used */
    /* only handle IP packets */
    if (* (LONG*) ecb->ECB_ProtocolID != 0 ||
        * (WORD*) (ecb->ECB_ProtocolID + 4) != htons(0x0800))
        return 1;

#ifdef LOG_ECB_ACTIVITY
    if (LogECBHandle) {
        int TGID = SetThreadGroupID(DPC_TGID);
        LogMsg(LogClientHandle, LogECBHandle, FALSE,
            "TINET Enqueue(%08lx)\n", ecb);
        SetThreadGroupID(TGID);
    }
#endif /* LOG_ECB_ACTIVITY */
    Enqueue(&NewQ, ecb);
    return 0;
}

LONG InetControl(void) {
    return 0xfffff81;
}

void ClearConnection(WORD port) {
    if (ScanBits(ConnectionMask, port, port+2) == port) {
        BitClear(ConnectionMask, port);
        --DIOSStats->TxOKMultipleCollisions;
    }
}

int AllocateConnection(WORD port) {
    if (ScanBits(ConnectionMask, port, port+2) != port) {
        /* see if there is a connection left */
        if (DIOSStats->TxOKMultipleCollisions < DPCMaxConnections) {
            /* allocate the new connection */
            BitSet(ConnectionMask, port);
            ++DIOSStats->TxOKMultipleCollisions;
            return 1;
        }
        return 0;
    }
    return 1;
}

LONG ConnectionLimiter(ECB* ecb, LONG board, void* chainID) {

```

```

BYTE* IPHeader = ecb->ECB_Fragment[0].FragmentAddress;
BYTE* TCPHeader = 0;
board = board;
chainID = chainID;
/* not used */
/* not used */

/* only handle IP packets */
if (* (LONG*) ecb->ECB_ProtocolID != 0 ||
    * (WORD*) (ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

/* double check stats, hopefully upper layer is kosher, but */
if (DIOSStats == 0) {
    releaseECB:
    --RxECBRTag->RxResourceCount;
    CLSLEFastSendComplete(ecb);
    return 0;
}

/* only check TCP packets to our interface */
if (IP_PROTO(IPHeader) != IPPROTO_TCP ||
    IP_DST_ADDR(IPHeader) != DPC_IP_Address)
    return 1;

TCPHeader = IPHeader + IP_HDR_LEN(IPHeader) * 4;
if (ecb->ECB_Fragment[0].FragmentLength < (TCPHeader + 20) - IPHeader)
    return 1;

if (TCP_CODE(TCPHeader) & (TCP_FIN|TCP_RST)) {
    /* release the connection */
    ClearConnection(ntohs(TCP_DST_PORT(TCPHeader)));
}
else if (TCP_CODE(TCPHeader) & TCP_SYN) {
    /* allocate the connection */
    if (!AllocateConnection(ntohs(TCP_DST_PORT(TCPHeader))))
        goto releaseECB;
}
return 1;

/* IP Manipulation */
# if 0
char *chksum (BYTE *buf, unsigned cnt)
{
    static unsigned char crc_bytes[2];
    BYTE rbl;
    WORD rax, rcx;
    int redx;
    BYTE *rdssi;

    crc_bytes[0] = crc_bytes[1] = 0;

    rcx = cnt;
    rdssi = buf;
    rbl = rcx;
    rcx = rcx >> 1;
    radx = 0;
    if (rcx != 0)
    {
        while (rcx--)
        {
            rax = * (WORD *) rdssi;
            rdssi += 2;

```

```

if (redx & 0xffff0000)
    redx++;
redx &= 0x0000ffff;
redx += rax;
}
if (redx &= 0x0000ffff)
{
    redx &= 0x0000ffff;
    redx++;
}
if (rbl & 1)
{
    rax = 0;
    rax = *rdssi;
    redx += rax;
    if (redx &= 0x0000ffff)
        redx++;
}
redx = ~redx;
crc_bytes[0] = redx & 0xff;
crc_bytes[1] = (redx >> 8) & 0xff;
return (char *)crc_bytes;
}

#endif

#ifdef __GNUC__
/*
 * This is a version of ip_compute_csum() optimized for IP headers, which
 * always checksum on 4 octet boundaries.
 * This version is constructed from various places in the linux and Hughes
 * sources.
 */

static inline unsigned short ip_fold_lcomp_csum(unsigned long sum) {
    __asm__ (
        "movl %w1, %w0\n\t"
        "shrl $16, %1\n\t"
        "addw %w1, %w0\n\t"
        "adcw $0, %w0\n\t"
        "notw %w0"
        : "a" (csum)
        : "b" (sum));
    return csum;
}

static inline unsigned short ip_fast_csum(unsigned short *buff, int wlen) {
    unsigned long sum = 0;

    if (wlen) {
        unsigned long eax;
        /* Suggested speedup:
        1:
        movl (%esi), %ebx
        lea (%esi+4), %esi
        adcl %ebx, %eax
        decl %ecx
        jnz 1b
        adcl $0, %eax
        movl %eax, %ebx
        shrl $16, %eax
        addw %ebx, %eax
        adcl $0, %eax
        */
    }

```

```

xorl $0xffff, %eax
*/
__asm__ ("crl\n"
        "1:\n\t"
        "lodsl\n\t"
        "adcl $3, %0\n\t"
        "loop 1b\n\t"
        "adcl $0, %0\n\t"
        : "=r" (sum), "=S" (buff), "=c" (wlen), "=a" (eax)
        : "0" (sum), "1" (buff), "2" (wlen));
    }
    return ip_fold_lcomp_csum(sum);
}

#define chksum(b, l) ip_fast_csum(b, (l) / 4)

static inline unsigned short ip_adjust_csum(unsigned short oldsum,
        unsigned short oldval,
        unsigned short newval) {
    unsigned long sum = ((unsigned short)~oldsum);
    sum += ((unsigned short)~oldval);
    sum += newval;
    return ip_fold_lcomp_csum(sum);
}

#endif /* __GNUC__ */

static int DummyFrame(FRAG_DESC* frag) { /* not used */
    frag = frag;
    return 0;
}

int (*DPCDropFrame)(FRAG_DESC* frag) = DummyFrame;

void FilterQueue(void* arg) {
    ECB* ecb;
    ECB* rover;
    BYTE* IP;
    BYTE* TCP;
    int excess;
    arg = arg; /* not used */

    RenameThread(GetThreadId(), "DPCAgent Filter");
    for (;;) {
        if (ExitingFlag)
            return;
        TimedWaitOnLocalSemaphore(NewQ.semaphore, 1000);
        if (!NewQ.head)
            continue;

        ecb = Dequeue(&NewQ);
        ecb->activityTimer = milliclock();
        IP = ecb->ECB_Fragment[0].FragmentAddress;

        if (DIOSStats == 0) {
            releaseECB:
            DPCDropFrame((FRAG_DESC*)&ecb->ECB_FragmentCount);
            --TxECBRTag->RTResourceCount;
            CUSLPastSendComplete(ecb);
            #ifdef LOG_ECB_ACTIVITY
            FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                "TINET Release(%08lx)\n", ecb));
            #endif
        }
    }
}

```



```

BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
BYTE* roverTCP;
excess = (rover->ECB_Fragment[0].FragmentLength -
  IP_HD_LEN(roverIP) * 4);
if (excess > 0) {
  roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
}
else {
  roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
  excess += rover->ECB_Fragment[1].FragmentLength;
}
if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
  excess >= 20 &&
  (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
  IP_PROTO(roverIP) == IPPROTO_TCP &&
  IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
  TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
  IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
  TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
  TCP_CODE(roverTCP) & TCP_ACK &&
  (htonl(TCP_ACKNUM(roverTCP)) + htons(TCP_WINDOW(roverTCP)) <
  htonl(TCP_ACKNUM(TCP)) + htons(TCP_WINDOW(TCP))) {
  /* move ACK information over to TxQ and release this packet */
  TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    TCP_WINDOW(roverTCP),
    TCP_WINDOW(TCP));
  TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    (WORD)TCP_ACKNUM(roverTCP),
    (WORD)TCP_ACKNUM(TCP));
  TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    TCP_ACKNUM(roverTCP)>>16,
    TCP_ACKNUM(TCP)>>16);
  TCP_ACKNUM(roverTCP) = TCP_ACKNUM(TCP);
  TCP_WINDOW(roverTCP) = TCP_WINDOW(TCP);
  ++DIOSStats->TxAbortExcessCollisions;
  goto releaseECB;
}
goto enqueueTxQ;
goto enqueueTxQ;

filterUDP:
{
  BYTE* UDP = TCP;
  BYTE* DNS;

  /* ECB contents determined by inspection, there are safer methods */
  if (excess < 8)
    goto enqueueTxQ;

  /* filter DNS only */
  if (UDP_DST_PORT(UDP) != htons(53))
    goto enqueueTxQ;

  excess -= 8;
  DNS = (excess > 0) ? (UDP + 8) : ecb->ECB_Fragment[1].FragmentAddress;

  for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
    BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
    BYTE* roverUDP;
    BYTE* roverDNS;
    excess = (rover->ECB_Fragment[0].FragmentLength -
      IP_HD_LEN(roverIP) * 4);
    if (excess > 0) {
      roverUDP = roverIP + IP_HD_LEN(roverIP) * 4;

```

```

    }
    else {
      roverUDP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
      excess += rover->ECB_Fragment[1].FragmentLength;
    }
    if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
      excess >= 8 &&
      (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
      IP_PROTO(roverIP) == IPPROTO_UDP &&
      IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
      UDP_DST_PORT(roverUDP) == UDP_DST_PORT(UDP) &&
      IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
      UDP_SRC_PORT(roverUDP) == UDP_SRC_PORT(UDP) &&
      (roverDNS = ((excess - 8) > 0) ?
        (roverUDP + 8) :
        (rover->ECB_Fragment[1].FragmentAddress)) &&
      *((LONG*)DNS) == *((LONG*)roverDNS) {
      ++DIOSStats->TxAbortLateCollision;
      goto releaseECB;
    }
    goto enqueueTxQ;
  }
}

/* SLIP, PPP, Modem Manipulation */
#define MAX_READ_BUF 128

int InetState = MODEM_IDLE;
static BYTE SlipEndpkt[1] = (END);

int WaitingLines = 0, NextWait = 0;
char WaitingBuffer[MAX_READ_BUF];
int WaitingIndex = 0;
LONG ConnectingTimeout = 0;
LONG ConnectingRedial = FALSE;

int BaudRate[] =
{
  2400, /* 0 */
  3600, /* 1 */
  4800, /* 2 */
  7200, /* 3 */
  9600, /* 4 */
  19200, /* 5 */
  38400, /* 6 */
  57600, /* 7 */
  115200 /* 8 */
};

void InitLogin()
{
  int i;
  char *nextWait;

  WaitingLines = 0;
  if (!DioCfg.auto_login)
  {
    WaitingIndex = 0;
    WaitingBuffer[WaitingIndex] = '\0';
    NextWait = 0;
  }
}

```

```

ConnectingTimeout = 0;
for (i = 0, nextWait = DloCfg.wait_for_1; i < 9; i++, nextWait +=
sizeof(DloCfg.wait_for_1))
{
    if (*nextWait)
        WaitingLines++;
}

static BYTE MTUBuffer[8192];

int SLIPSendRoutineOpt(FRAG_DESC* fragStruc)
{
    LONG count = 0;
    BYTE* output = MTUBuffer;

    *output++ = END;
    while (count < fragStruc->FragmentCount)
    {
        FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
        BYTE* frame = (BYTE*)frag->FragmentAddress;
        LONG length = frag->FragmentLength;

        while (length-- > 0)
        {
            switch (*frame)
            {
                case END:
                    *output++ = ESC;
                    *output++ = ESC_END;
                    break;
                case ESC:
                    *output++ = ESC;
                    *output++ = ESC_ESC;
                    break;
                default:
                    *output++ = *frame;
                    break;
            }
            if (header == 0x80000000)
                header = (*frame & 0x0f) * 4;
            if (--header == 0)
                dataStart = output;
            ++frame;
        }
        ++count;
    }

    if (output - MTUBuffer < 20 ||
        output - MTUBuffer > DloGetWriteBufferSize())
        return 0;
    DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
    DloSend(MTUBuffer, dataStart - MTUBuffer, DLO_INET_TIMEOUT);
    DloSend(dataStart, output - dataStart, DLO_INET_TIMEOUT);
    DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
    return 1;
}

int (*DPCtxFrame)(FRAG_DESC* fragStruc) = SLIPSendRoutineOpt;

/*****
*
* IPSendRoutine(ECB *tcb)
*
* Description:
*
* Input:   tcb
*          int Control Block
*
* Output:  nothing
*
* Returns: 0 if finished with ECB
*
*****/
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
    /* version 4, length 5 */
    /* tos */
    /* length */

```

```

0, 0,
0, 0,
0x7f,
4,
);
#define IPHeaderIdent ((WORD*)&IPHeader[4])
int
{
    FRAG_DESC* fragStruc = alloca(sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * (
    ecb->ECB_FragmentCount + 2)));
    WORD frame_size = ecb->ECB_DataLength;
    int options_collapsed = 1;
    LONG currFrag = 0;
    BYTE* ecbIPHeader = ecb->ECB_Fragment[0].FragmentAddress;

    /* initialize the copy of the tcb fragStruc */
    memcpy(fragStruc,
        &ecb->ECB_FragmentCount,
        sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * ecb->ECB_FragmentCount));

    if (frame_size < DloCfg.mtu &&
        {
            TUNNEL_ONLY_TCP
            /* UDP doesn't need tunnel header */
            (ecbIPHeader[9] != IPPROTO_TCP) ||

            /* either do "routed" packets */
            ((LONG*)&ecbIPHeader[12] != DPC_IP_Address)))
        goto skipFragger;

    memset(&fragStruc->FragmentDesc[fragStruc->FragmentCount],
        0,
        sizeof(FRAGMENTSTRUCT) * 2);

    frame_size += IP_TUNNEL_SIZE;
    /* fill IPHeader with tunnel data, including IP/gateway addresses,
     * and prepend to frag list.
     */
    *(WORD*)&IPHeader[2] = htons(frame_size);
    ++IPHeaderIdent;
    *(WORD*)&IPHeader[10] = 0; /* checksum, for now */
    *(LONG*)&IPHeader[12] = DloCfg.ip_address;
    *(LONG*)&IPHeader[16] = DloCfg.gateway_address;
    memcpy(fragStruc->FragmentDesc + 1,
        fragStruc->FragmentDesc,
        sizeof(FRAGMENTSTRUCT) * fragStruc->FragmentCount);
    fragStruc->FragmentDesc[0].FragmentAddress = IPHeader;
    fragStruc->FragmentDesc[0].FragmentLength = IP_TUNNEL_SIZE;
    ++fragStruc->FragmentCount;
    ++currFrag;
    *(WORD*)&IPHeader[10] = chksum((WORD *)IPHeader,
        IP_TUNNEL_SIZE);

    while (frame_size > DloCfg.mtu)
    {
        /*
         * Shucks. Have to fragment the packet.
         * This algorithm is roughly per RFC791.
         */
        LONG OIHL = fragStruc->FragmentDesc[0].FragmentLength;
        BYTE OMF = IPHeader[6] & 0x20;
        LONG NFB (DloCfg.mtu - OIHL) & 0xf1f8;
        WORD TL = OIHL + NFB;

```

```

IPHeader[6] |= 0x20; /* set More Fragments */
*(WORD*)&IPHeader[2] = htons(TL);
*(WORD*)&IPHeader[10] = 0; /* clear checksum */
*(WORD*)&IPHeader[10] = chksum((WORD *)IPHeader, OIHL);

/*
 * Now fake out the fragStruc to reflect TL.
 * Hang on to enough information to remove the TL less OIHL
 * later.
 */
TL -= OIHL;
frame_size -= TL;
while (TL > 0 &&
    fragStruc->FragmentDesc[currFrag].FragmentLength <= TL)
{
    TL -= fragStruc->FragmentDesc[currFrag].FragmentLength;
    ++currFrag;
}
if (TL > 0)
{
    /*
     * This frag gets split into 2 pieces.
     */
    memcpy(fragStruc->FragmentDesc + currFrag + 1,
        fragStruc->FragmentDesc + currFrag,
        sizeof(FRAGMENTSTRUCT) *
            (fragStruc->FragmentCount - currFrag));
    ++fragStruc->FragmentCount;
    fragStruc->FragmentDesc[currFrag].FragmentLength = TL;
    ++currFrag;
    fragStruc->FragmentDesc[currFrag].FragmentLength -= TL;
    fragStruc->FragmentDesc[currFrag].FragmentAddress = ((ch
    ar*)fragStruc->FragmentDesc[currFrag].FragmentAddress) + TL;
}
TL = fragStruc->FragmentCount - currFrag + 1;
fragStruc->FragmentCount = currFrag;

if (DPCTXFrame(fragStruc) == 0)
    return 0;

if (!options_collapsed) {
    LONG offset = 20;
    while (offset < OIHL && IPHeader[offset]) {
        if (IPHeader[offset] & 0x80) /* copy */
            offset += IPHeader[offset + 1];
        else /* collapse */
            LONG len = IPHeader[offset + 1];
            memcpy(IPHeader + offset,
                IPHeader + offset + len,
                OIHL - (offset + len));
            OIHL -= len;
    }
    offset = fragStruc->FragmentDesc[0].FragmentLength;
    fragStruc->FragmentDesc[0].FragmentLength = (OIHL + 3) &
        0x3c;
    memset(IPHeader + OIHL,
        0,
        fragStruc->FragmentDesc[0].FragmentLength - OIHL);
    IPHeader[0] = 0x40 | (fragStruc->FragmentDesc[0].Fragmen
    tCount / 4);
    frame_size -= offset - fragStruc->FragmentDesc[0].Fragme
    ntLength;
    options_collapsed = 1;
}

```

```

/* Adjust the frag list to "remove" the fragment just sent.
 */
memmove(fragStruc->FragmentDesc + 1,
        fragStruc->FragmentDesc + currFrag,
        sizeof(FRAGMENTSTRUCT) * TL);
fragStruc->FragmentCount = TL;
currFrag = 1;

/* compute new IPHeader values: fragment offset */
*(WORD*)&IPHeader[6] = htons((ntohs(*(WORD*)&IPHeader[6])) &
                                0xffff)
    + (NFB / 8));

IPHeader[6] |= OMF;
if (frame_size <= DloCfg.mtu)
{
    *(WORD*)&IPHeader[2] = htons(frame_size);
    *(WORD*)&IPHeader[10] = 0; /* clear checksum */
    *(WORD*)&IPHeader[10] = chksum((WORD *)IPHeader,
                                   fragStruc->FragmentDesc
                                   [0].FragmentLength);
    break;
}

skipFragger:
/* send the remaining (possibly ALL) portion of the frame */
if (DPCTxFrag(fragStruc) == 0)
    return 0;

if (DIOStats)
{
    ++DIOStats->TotalTxPacketCount;
    if ((DIOStats->TotalTxOKByteCountLow += ecb->ECB_DataLength) <
        ecb->ECB_DataLength)
        ++DIOStats->TotalTxOKByteCountHigh; /* wrapped */
    return 1;
}

static void EmptyESR(ECB* ecb) {
}

unsigned char RawEnvelope[14] = {
    0x00, 0x00, 0x0c, 0x0a, 0x0b, 0x0c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x08, 0x00,
};

ECB RawECB = {
    0,
    /* ECB_NextLink */
    /* ECB_PreviousLink */
    /* ECB_Status */
    /* ECB_ESR */
    /* ECB_StackID */
    /* ECB_ProtocolID */
    /* ECB_BoardNumber */
    /* ECB_ImmediateAddress */
    { 0 },
    /* ECB_DriverWorkspace */
    { 0 },
    /* ECB_ProtocolWorkspace */
    /* ECB_DataLength */
    -1,
    /* ECB_FragmentCount */
    -1,
    /* RawEnvelope, sizeof(RawEnvelope) },
};

/* technically unsafe, but works */
FRAGMENTSTRUCT RawFrag[15] = {
    IPHeader, sizeof(IPHeader),
};

#define RawSendRoutineDeBug RawSendRoutineOpt
int RawSendRoutineOpt(FRAG_DESC* fragStruc) {
    int i;
    for (i = 20000; --i > 0; ) {
        if (RawECB.ECB_Status == 0)
            goto rawSend;
        if (ExitingFlag)
            return 0;
        ThreadSwitchWithDelay();
    }
    return 0;
rawSend:
    i = fragStruc->FragmentCount;
    RawECB.ECB_FragmentCount = i + 1;
    memcpy(RawFrag,
           fragStruc->FragmentDesc,
           i * sizeof(FRAGMENTSTRUCT));
    RawECB.ECB_DataLength = sizeof(RawEnvelope);
    while (i > 0)
        RawECB.ECB_DataLength += RawFrag[--i].FragmentLength;
    CLSLSendPacket(&RawECB);
    return 1;
}

void DisplayWaitStatus(void)
{
    char statusStr[80];
    char *waitingStr;

    if (WaitingLines == 0)
        return;

    waitingStr = (char *)DloCfg.wait_for_1[NextWait*30];
    NWSprintf(statusStr, MSG("Modem Status: Waiting for \"%s\\\"n", 557), wait
    ingStr);
    UpdateModemStr(statusStr);
}

void InetStateChange(int state) {
    if (DloCfg.out_protocol == OUT_NETWORK) {
        InetState = PROTOCOL_CONNECTED;
        return;
    }
    switch (state) {
        case DLOS_IDLE:
        case DLOS_DISC_1:
        case DLOS_DISC_2:
        case DLOS_DISC_3:
        case DLOS_DISC_4:
            InetState = MODEM_IDLE;
            if (ConnectingRedial) {
                DloEndConn();
                ConnectingRedial = FALSE;
            }
            break;
        case DLOS_CONN:
            InetState = MODEM_CONNECTED;
            InitLogin();
            if (InetAsleep)
                ResumeThread(ppcInetPID);
            break;
    }
}

```



```

default:
break;
)

)

/******
* FUNCTION: Convert Internet Address
* DESCRIPTION: converts a character string containing the Internet address
* into a form that BIC DD understands.
* e.g. 139.85.124.06 (8B.55.7C.06) into 067C58B0000
*****
void convert_address(char *lpszIpAddress)
{
    char *p;
    int i = 0;
    char tmp[20], tmp1[10];
    tmp[0] = 0;
    while((p=strchr(lpszIpAddress, (int)'.')) != NULL)
    {
        i = atoi(p+1);
        NWSprintf(tmp1, MSG("%02X", 477), i);
        strcat(tmp, tmp1);
        *p = 0;
    }
    i = atoi(lpszIpAddress);
    NWSprintf(tmp1, MSG("%02X", 478), i);
    strcat(tmp, tmp1);
    strcat(tmp, MSG("0000", 479));
    CStrCpy(lpszIpAddress, tmp);
}

MACAddr_t      HIAddr;
LONG            InetChannel;

void make_hi_key(chunk *key)
{
    int i;
    LONG sn;
    BYTE serialNum[9];
    BYTE serialNumPacked[3];
    BYTE x;

    DIOGetSN(serialNum);
    sn = atol(serialNum);
    NWSprintf(serialNum, MSG("%06lx", 480), sn);

    pack_mac_addr(serialNumPacked, 3, serialNum, 6);
    x = serialNumPacked[0];
    serialNumPacked[0] = serialNumPacked[2];
    serialNumPacked[2] = x;

    key->b[0] = serialNumPacked[0] ^ 0xff;
    key->b[1] = serialNumPacked[1] ^ 0xff;
    key->b[2] = serialNumPacked[2] ^ 0xff;
    for(i = 3; i < 8; i++)
        key->b[i] = 0x00 ^ 0xff;

    MACbuildAddr(serialNum, MAC_HI, 0, &HIAddr);
}

void InetChangeProtocol(void)
{
    switch (DloCfg.out_protocol) {
    case OUT_PPP:
        DPCTxFrame = DebugFlag ? PPpSendRoutineDebug : PPpSendRoutineOpt;
        break;
    case OUT_NETWORK: {
        void (*ControlEntryPoint)(void) = 0;
        struct DriverConfigurationStructure* dvrCfg = 0;
        if (CISLGetMLIDControlEntry(DloCfg.net_interface,
                                   &ControlEntryPoint))
        {
            goto skipDriver;
        }
        dvrCfg = (struct DriverConfigurationStructure *)
            CommandMlid(DloCfg.net_interface, 0, (LONG)ControlEntryP
            oint);
        memcpy(RawEnvelope + 6, dvrCfg->DNodeAddress, 6);
        skipDriver:
            RawECB.ECB_BoardNumber = DloCfg.net_interface;
            memcpy(RawEnvelope, DloCfg.net_addr, 6);
            DPCTxFrame = DebugFlag ? RawSendRoutineDebug : RawSendRoutineOpt;
            break;
        }
    case OUT_SLIP:
        DPCTxFrame = DebugFlag ? SLIPSendRoutineDebug : SLIPSendRoutineOpt;
        break;
    }
    InetStateChange(DLOS_DISC_4);
    DloEndConn();
}

int ProcessLogin(void)
{
    BYTE value;
    char *sendStr, *waitStr;
    char sendBuf[40];
    LONG nextTimeout;

    /* No use trying if we aren't even connected */
    /* Get out if we're done */

    if (WaitingLines == 0 || DloCfg.auto_login == FALSE)
    {
        return(TRUE);
    }

    if (!DloConnected())
        return(FALSE);

    /* Timeout if we've waited too long for this wait */
    if (ConnectingTimeout == 0)
    {
        ConnectingTimeout = GetCurrentTime() + DloCfg.wait_timeout * 1
        8;
    }

    if (GetCurrentTime() > ConnectingTimeout)

```



```

DPCGetIPAddress(&DPC_IP_Address);
ECBSignature);
if (DioCfg.out_protocol == OUT_NETWORK)
    InetState = PROTOCOL_CONNECTED;

mainloop:
    while (!ExitingFlag)
    {
        InetAsleep = TRUE;
        if (millidelay > 55)
            delay(millidelay);
        else if (millidelay > 0)
            ThreadSwitchWithDelay("LowPriority"/());
        else
            ThreadSwitch();
        InetAsleep = FALSE;

        while (DIOBoard && removedCount != DIORemovedCount)
        {
            BYTE address[8];
            BYTE szBicBCDAddress[20];
            struct DriverStatsStructure* stats = 0;
            LONG ip_address = ntohl(DioCfg.ip_address);

            removedCount = DIORemovedCount;
            /* Enable internet reception */

            /* Yuk. We'll change this later to get rid these extra s
            NWSprintf(szBicBCDAddress, MSG("%d.%d.%d.%d", 620),
                (ip_address >> 24) & 0xff,
                (ip_address >> 16) & 0xff,
                (ip_address >> 8) & 0xff,
                (ip_address) & 0xff);

            convert_address(szBicBCDAddress);
            if (!pack_mac_addr(address, 6,
                szBicBCDAddress, CStrLen(szBicBCDAddr

            {
                /* UpdateModemStr(MSG("ERROR: could not pack mac
                millidelay = 500;
                break;
            }

            /* Sending an esr address of -1 tells MLID to handle rec
            if (DIOOpenChannel(address,
                (int (*)())0xfffffffff,
                &InetChannel))
            {
                millidelay = 500;
                removedCount = (LONG)(-1);
                break;
            }
            if (ExitingFlag)
                break;
            DIOAddHIAddr(InetChannel, (BYTE *)&HIAddr);
            DPCGetMLIDStats(&stats);
            DioStats->TxOKMultipleCollisions = 0;
            if (CLSLRegisterPreScanTxChain(TxChainRtag,
                DIOBoard,
                3, /* next to last */
                &TxChainID,

```

```

InetQueuePacket,
InetControl,
TxECBRtag))

```

```

        millidelay = 500;
        removedCount = (LONG)(-1);
        break;
    }

```

```

    if (CLSLRegisterPreScanRxChain(RxChainRtag,
        DIOBoard,
        3, /* next to last */
        &RxChainID,
        ConnectionLimiter,
        InetControl,
        RxECBRtag))
    {

```

```

        millidelay = 500;
        removedCount = (LONG)(-1);
        break;
    }

```

```

    if (DioCfg.out_protocol == OUT_PPP &&
        InetState == PROTOCOL_CONNECTED)
    {
        PPPBackground();
    }

```

```

    if (TxQ.head == 0 &&
        (InetState <= MODEM_CONNECTING ||
         InetState >= PROTOCOL_CONNECTED))
    {
        TimedWaitOnLocalSemaphore(TxQ.semaphore, 200);
        millidelay = 0;
        continue;
    }

```

```

    switch (InetState)
    {
        case MODEM_CONNECTED:
            if (!ProcessLogin())
            {
                millidelay = 500;
                break;
            }
            InetState = LOGIN_CONNECTED;

```

```

        ; /* fallthru */
        case LOGIN_CONNECTED:
            if (!ConnectProtocol())
            {
                DioEndConn();
                millidelay = 15 * 1000;
                break;
            }
            InetState = PROTOCOL_CONNECTED;
            millidelay = 1;
            break;
        case PROTOCOL_CONNECTED:

```

```

        {
            LONG count = 0;
            if (DioCfg.out_protocol != OUT_NETWORK &&
                ALOWriteStatus(ATOPortHandle, &count, 0))
            {
                millidelay = 200;
                break;
            }
        }
    }
}

```

```

if (count == 0)
{
    LONG milliclock = milliclock();
    ECB* ecb;
    ecb = TxQ.head;
    millidelay = 100;
    while (ecb->activityTimer > milliclock)
    {
        LONG diff = ecb->activityTimer - milliclock;
        if (diff < millidelay)
            millidelay = diff;
        if ((ecb = ecb->ECB.NextLink) == 0)
            goto mainloop;
    }
    Remove(&TxQ, ecb);
    if (ecb->activityTimer < milliclock() - 60000) {
        if (ecb->activityTimer)
            ++DIOStats->TxAbortExDeferral;
    }
    else
        IPSendRoutine(ecb);
    ReleaseECB(ecb);
    if (DloCfg.out_protocol != OUT_NETWORK)
        AIOWriteStatus(AIOPortHandle, &count, 0);
}
millidelay = (count *
10 * /* Tx bits with framing */
1000 / /* milliseconds */
BaudRate[DloCfg.tinet_baud_index]);
break;
}
case MODEM_IDLE:
    if (nextStartConn < time(0))
    {
        InitLogin();
        DloStartConn(DLO_INET_TIMEOUT);
        InetState = MODEM_CONNECTING;
        nextStartConn = time(0) + 30;
        /* fallthru */
    }
    default:
        millidelay = 10 * 1000;
        break;
    }
}

DIOCloseChannel(InetChannel);

CLSLDeRegisterPreScanRxChain(RxChainID);
CLSLDeRegisterPreScanTxChain(TxChainID);
while (TxQ.head)
    ReleaseECB(Dequeue(&TxQ));
while (NewQ.head)
    ReleaseECB(Dequeue(&NewQ));
CloseLocalSemaphore(TxQ.semaphore); TxQ.semaphore = 0;
CloseLocalSemaphore(NewQ.semaphore); NewQ.semaphore = 0;
UnregisterForEvent(protocolBindHandle);

DPCInetPID = 0;
return;
}

```